uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería en Tecnologías de la Telecomunicación
2018-2019

Trabajo Fin de Grado

"DISEÑO DE UN SISTEMA DE CONTROL PARA EXPERIMENTOS DE RADIACIÓN DE CIRCUITOS BASADO EN RASPBERRY-PI"

Adrián Alcalde Albert

Tutor/es

Mario García Valderas

1 Julio, Universidad Carlos III de Madrid

RESUMEN

En este documento se presenta el diseño de un sistema de control para dispositivos que se utilizan en experimentos con radiación. Este sistema se ayuda de una Raspberry Pi.

El objetivo que se persigue en este documento es el de mejorar los diseños que previamente se han utilizado para este tipo de experimentos. En este sentido, se describe el diseño y la fabricación del sistema de control, así como, la implementación de la Raspberry Pi y las comunicaciones hardware y software pertinentes con los dispositivos expuestos a radiación en el experimento.

Por último, se presentan las pruebas justificadas de que este nuevo diseño propuesto es una mejora sustancial respecto a los sistemas que se han utilizado en anteriores experimentos. Con estas se detallan las fortalezas y las debilidades de esta nueva implementación, así como la batería de pruebas que se han realizado para comprobar su robustez y funcionamiento.

Palabras clave: sistema de control, radiación, Zybo, Raspberry Pi.

AGRADECIMIENTOS

Por último, no puedo dejar de lado agradecer a todas y cada una de las personas que me han acompañado en este largo y duro camino.

En primer lugar, quiero centrar mis agradecimientos a mis padres, quienes me han apoyado y han sufrido cada día desde el comienzo y hasta el final, compartiendo cada éxito y cada fracaso durante toda esta etapa. Gracias a su apoyo, a su dedicación y a su cariño, que han hecho más llevadero este proceso cada día.

En segundo lugar, dar mi agradecimiento a mi hermana, quien ha tenido que soportar también este largo proceso y me ha apoyado durante la elaboración de este trabajo.

Por supuesto, dar también las gracias a mi pareja. Por su apoyo, por sus medidas contra mis agobios, por su gran cariño y por contribuir con su ayuda a mejorar la elaboración de este trabajo.

Gracias a mis amigos, quienes me han visto menos en este tiempo, por estar centrado en el proceso del trabajo, pero los cuales han estado siempre ahí cuando los he necesitado.

Agradecer también a mi tutor, quien me ha ayudado, acompañado y enseñado en la realización de este trabajo.

Y, por último, quiero hacer especial mención a mi hermano, quien me ha dado el principal apoyo para llegar a donde he llegado, para empezar y para terminar este trabajo.

Muchísimas gracias.

Contenido

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 INTRODUCCIÓN

1.1 Antecedentes

La premisa inicial parte del interés de estudiar el comportamiento de los circuitos cuando se radian sobre ellos con protones y neutrones.

Los neutrones al tener una masa grande atraviesan los circuitos produciendo agujeros en las pistas lo puede provocar cambios a nivel lógico donde bits pasan de ‘1’ a ‘0’, y viceversa.

En cambio, los protones tienen mucha menos masa y no son capaces de atravesar los circuitos. Al igual que los neutrones, pueden producir agujeros en las pistas lo puede provocar cambios a nivel lógico donde bits pasan de ‘1’ a ‘0’, y viceversa.

El primer escenario se va a llevar a cabo la realización de un experimento en el que se radian neutrones y protones.

Este experimento se realiza dentro de un búnker de radiación en el que se dispone de una máquina capaz de radiar a través de un haz protones y neutrones. En el camino de este haz se encuentran varias placas, los dispositivos a radiar. A unos dos metros se encuentra dos placas que actúa de sistema de control del experimento. Al otro lado de la sala, situada a 15m, se ubica el puesto de control donde se sitúan los técnicos que a través un ordenador se comunican con el sistema de control mencionado.

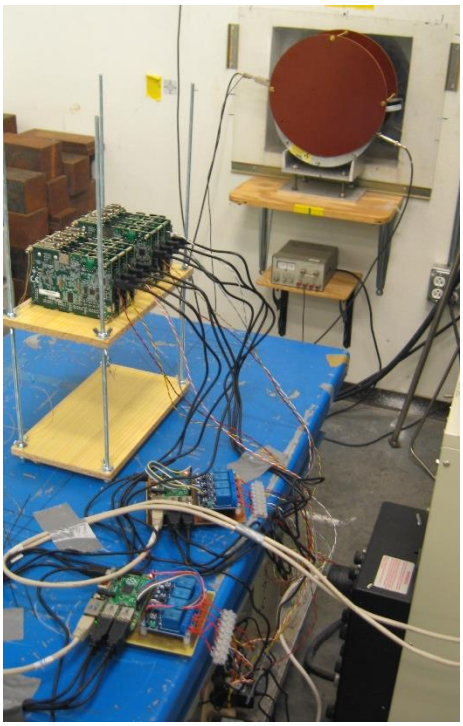


Fig. 1.1. Primer montaje del experimento con radiación

Los dispositivos expuestos durante el experimento son unas FPGA llamadas Zybo. Se dispones de 8 de ellas. Cada una se conecta dos formas: a través de un par de cables de alimentación por Zybo y a través de un cable USB para la comunicación.

Por cada cuatro Zybo se dispone de un sistema de control. El sistema de control está formado por una Raspberry Pi y un módulo de cuatro relés. La alimentación de este proviene del transformador de la Raspberry Pi. Además, la Raspberry Pi se conecta a través de un cable Ethernet que sale al exterior del búnker de radiación y se conecta al ordenador que se encuentra en el puesto remoto. Desde este se controla el estado de las Zybo y el sistema de control a través de una conexión SSH.

Durante el experimento, las Zybo se comunican a través del cable USB enviando datos del su comportamiento. Estos datos los recibe la Raspberry Pi y los almacena en un archivo log por cada Zybo. Cuando la Zybo detecta se ha producido un error envía un aviso a la Raspberry Pi y esta utiliza el módulo de relés al que está conectada para reiniciar la alimentación de la Zybo.

Como se observa, por cada montaje Zybo-Sistema de control supone un total de 4 transformadores, con 8 pares de cable de alimentación de la Zybo, 4 cables USB y un cable Ethernet. A esto se le añade 5 cables para conectar los pines GPIO de la Raspberry Pi y su propia alimentación. En definitiva, se utilizan una gran cantidad de cables lo que hace más complejo y tediosos los montajes.

Por lo mencionado anteriormente en el comportamiento de neutrones y protones, el montaje del experimento con estos últimos es igual que el experimento con neutrones, pero solo se puede radiar un solo dispositivo.

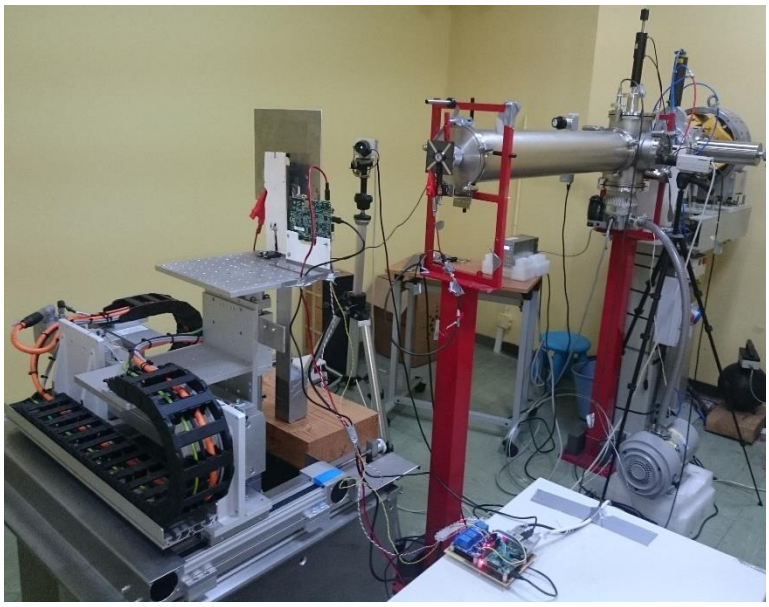


Fig. 1.2. Primer montaje del experimento con radiación (protones)

Con el objetivo de mejorar la complejidad, se realiza un segundo montaje en donde ahora el módulo de relés no se sitúa al lado de la Raspberry Pi, sino que es un módulo 'hat' para la Raspberry Pi, es decir, el módulo se ubica por encima de esta. Con este montaje al ir directamente conectado al 'pinout' de este, nos ahorramos únicamente los cables que se utilizan en los pines de la GPIO para controlar el módulo de relés. Es verdad que el sistema de control se hace más compacto, pero seguimos teniendo una gran cantidad de cables.

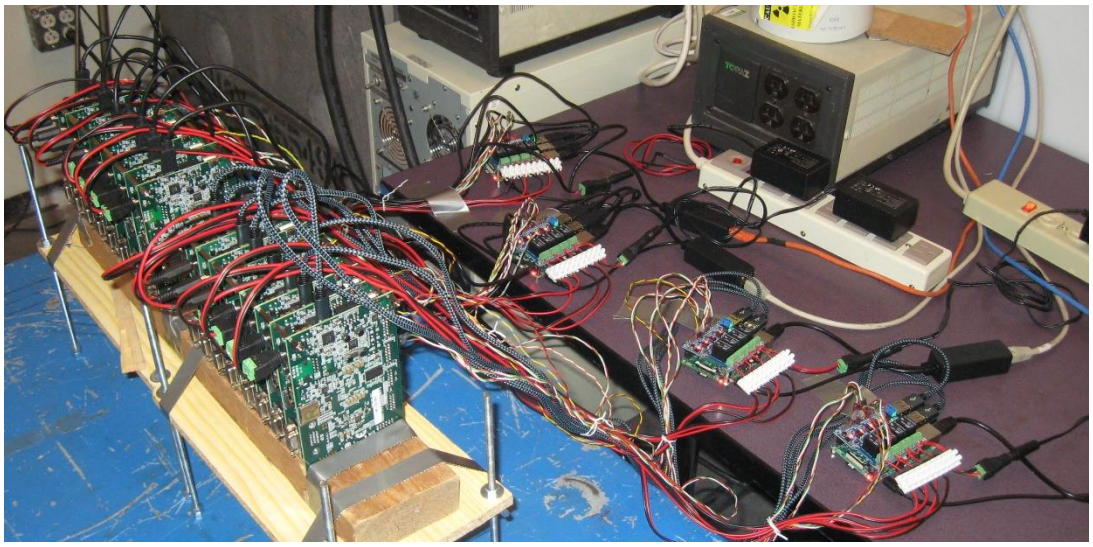


Fig. 1.3. Segundo montaje del experimento con radiación (neutrones)

A esto se le suma que al estar los transformadores dentro del laboratorio con radiación, si se produce algún mal funcionamiento en la Raspberry Pi, se detiene el intercambio de información y por consecuencia se debe entrar para desenchufar la Raspberry Pi para que se reinicie. Esto supone una gran pérdida de tiempo ya que se tiene que parar completamente el experimento y esperar a que desaparezca la radiación de la sala para

entrar con niveles de radiación seguros. Esto es poco eficiente ya que un laboratorio de este tipo su coste es por horas.

En conclusión, se necesita que el experimento esté funcionando de manera autónoma durante varios días sin necesidad de acceder a la sala y este montaje no es lo suficientemente eficiente para lo requerido.

1.2 Objetivos del proyecto

En este proyecto se pretende mejorar las desventajas que se han encontrado en los montajes que se han utilizado en los experimentos.

Para ello, se decide elaborar un nuevo diseño del sistema de control adaptándolo a nuestras necesidades, principalmente sea capaz de reducir al mínimo el cableado utilizado y que sea de funcionar como un puesto remoto junto con un ordenador de modo que pueda recuperarse ante fallas sin necesidad de acceder al laboratorio en el que se realiza el experimento.

Por otra parte, el hacer un diseño propio nos permite añadir nuevas funcionalidades para el futuro y crear una base para nuevas versiones del sistema de control.

1.3 Descripción

Como se ha mencionado en los objetivos del proyecto, se pretenden solucionar dos problemas principales. El primero es la gran cantidad de cableado que se utiliza y el segundo es la accesibilidad para reiniciar los dispositivos.

En este proyecto se van a distinguir dos partes, una parte de diseño hardware y otra de diseño software.

En cuanto al hardware, el nuevo diseño del sistema de control que se va a llevar a cabo se diferencian dos partes: una placa que se fabricará con una serie de componentes en sustitución del módulo de relés y una Raspberry Pi, que será el centro de procesamiento del sistema de control.

Ante el primer problema, se decide llevar a cabo como solución la implementación de un sistema único de alimentación. Esto significa que todos los componentes y dispositivos utilizados estarán alimentados por una fuente de tensión común.

En relación con lo anterior, se conectará a la placa una tensión de 48V que proviene de un switch PoE que se ubica fuera del búnker con radiación. Sin embargo, el problema que surge es que proporciona una tensión demasiado alta que dañaría el sistema de control. Por eso se necesita de una etapa intermedia para trabajar con tensiones de entorno a los 5V que sí son admisibles por parte de los dispositivos y la Raspberry Pi.

En este sentido, se debe incluir un conector de 40 pines donde se conecta la Raspberry Pi de un cable de 40 pines. Por ello, el objetivo es proporcionar a través de ellos la alimentación, así como incluir una serie de conectores para alimentar un máximo de cuatro dispositivos.

Por otro lado, como se ha mencionado anteriormente, el otro problema presente es la imposibilidad de la entrada de los técnicos a una sala con altos niveles de radiación y la ineficiencia de parar el experimento cada vez que se produzca un fallo que implique desconectar la Raspberry Pi o todos los dispositivos. Para solucionar este problema se decide implementar varias soluciones.

Una de las soluciones, como en diseños previos, es implementar una serie de relés para controlar la alimentación de las Zybo. Sin embargo, la diferencia radica, en que a través del conector de 40 pines que se ha establecido anteriormente, se pueden aprovechar lo pines GPIO tanto para controlar los relés, como para controlar una lógica que permita reiniciar la Raspberry Pi interrumpiendo su alimentación. Con esta funcionalidad no es necesario desenchufar de la red la fuente de alimentación del sistema de control para reiniciarla.

En este sentido, para comunicarnos con la Raspberry Pi para su control, se conecta un cable Ethernet a la Raspberry Pi que llega hasta el exterior del laboratorio en donde se conecta al puerto Ethernet del switch PoE, esto completaría la funcionalidad de puesto remoto que se requiere.

Por otro lado, aprovechando que a través el conector de 40 pines se puede disponer de todas las funcionalidades del 'pinout' de la Raspberry Pi. Se decide implementar varios conectores de pines para aprovechar los distintos protocolos serie que soporta la Raspberry Pi para la comunicación con los dispositivos. Estos protocolos son USB-UART, SPI, I2C Y UART.

Por último, la placa deberá disponer de una serie de leds que indiquen el funcionamiento de los dispositivos y la alimentación de esta.

En lo que respecta al software, se deben implementar distintos protocolos de comunicación para el intercambio de datos entre la Zybo y la Raspberry Pi del sistema de control. Estos protocolos, como se ha mencionado anteriormente, son USB-UART, SPI, I2C y UART.

En el protocolo USB-UART las Zybo se deben conectar mediante un cable USB desde su puerto mini USB al puerto USB de la Raspberry Pi. Los datos enviados en tramas se transmiten gracias a un programa que implementa una lógica para utilizar la interfaz UART de la Zybo. Este programa, ante la imposibilidad de disponer de una sala en la que haya dispuesta una máquina que radie protones o neutrones, incluye una lógica que simula los errores que se producen en la Zybo por el efecto de la radiación. El funcionamiento básico de este software consiste en introducir errores aleatorios que hace fallar a la Zybo tal y como podría suceder si estuvieran expuestas a un haz de protones o neutrones. Esto nos permite diseñar una interfaz en la Raspberry Pi capaz de controlar a las Zybo dependiendo de lo datos que reciba, sin necesidad de radiarla y sin necesidad manipularla más allá de ejecutar el programa. Estos datos se recopilan y se guardan en la Raspberry Pi mediante archivos logs. En este sentido, el protocolo UART también es compatible con esta lógica.

En consonancia con lo anterior, en el protocolo SPI e I2C se debe implementar un programa que sea capaz de intercambiar información básica, es decir, que sea capaz de enviar una serie de bytes y recibir los mismos bytes que se han enviado.

Por último, se enumeran de manera en detalles los principales requisitos establecidos:

- Encendido y apagado remoto de 4 dispositivos por medio de relés.
- Encendido y apagado remoto de la Raspberry Pi.
- Conversión de tensión de 48V a 5V para alimentar el sistema de control y los dispositivos.
- Alimentación de la Raspberry Pi a través de su 'pinout'.
- Ancho de la placa igual que la Raspberry Pi.
- Uso de agujero pasante en los componentes de la placa.
- Uso de SMD para leds y resistencias.
- Implementación de los conectores para la alimentación de los dispositivos y la comunicación.
- Implementación de un conector de 40 pines para el 'pinout' GPIO de la Raspberry Pi.
- Implementación del software de comunicación SPI/USB-UART/I2C, tanto en la Raspberry Pi como en la Zybo.

El montaje final de acuerdo con los requisitos establecidos se muestra en la siguiente figura.

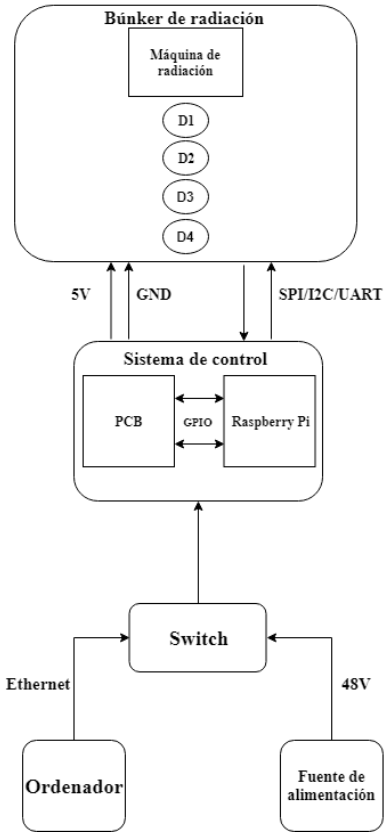


Fig. 1.4. Montaje de la solución propuesta

2 RASPBERRY PI

2.1 Introducción

Raspberry Pi es un ordenador de tamaño reducido (SBC) desarrollado por la Raspberry Pi Fundation. Se utiliza para aprender a programar y construir proyectos de electrónica, y para muchas de las cosas que hace un PC de escritorio, como hojas de cálculo, procesamiento de textos, navegación en Internet y juegos. [1]

Se pueden conectar distintos periféricos externos, tales como pendrives, ratones, teclados, pantalla o monitor, altavoces, entre otros.

Todas las Raspberry Pi poseen una serie de características comunes:

- HDMI.
- Mini Jack de audio 3.5mm.
- Conector DSI de pantalla.
- Conector CSI de cámara.
- Puerto mini USB de alimentación.
- 4x puertos USB.
- GPIO 40-Pin/26-Pin.
- Ethernet.

Existen distintos modelos de Raspberry Pi, los más conocidos son: Raspberry Pi B, Raspberry Pi B+, Raspberry Pi 2, Raspberry Pi 3B y Raspberry Pi 3B+.[2]

La Raspberry Pi posee también un lector de tarjetas donde se aloja la microSD que es el soporte donde se contiene el Sistema Operativo que carga al iniciarse.

Se han escogido dos modelos para las pruebas:

- **Raspberry Pi B+**: es un modelo actualizado de las Raspberry Pi B. En este último modelo se aumenta el tamaño de la placa como consecuencia de un mayor número de componentes en ella. Se ha cambiado el 'pinout' de 26 pines GPIO a 40 pines GPIO, se han incluido un total de 4 puertos USB frente a uno del modelo B, también ha sido mejorado el conector de salida de audio. Todo esto se acompaña con una sustancial mejora energética en su consumo.

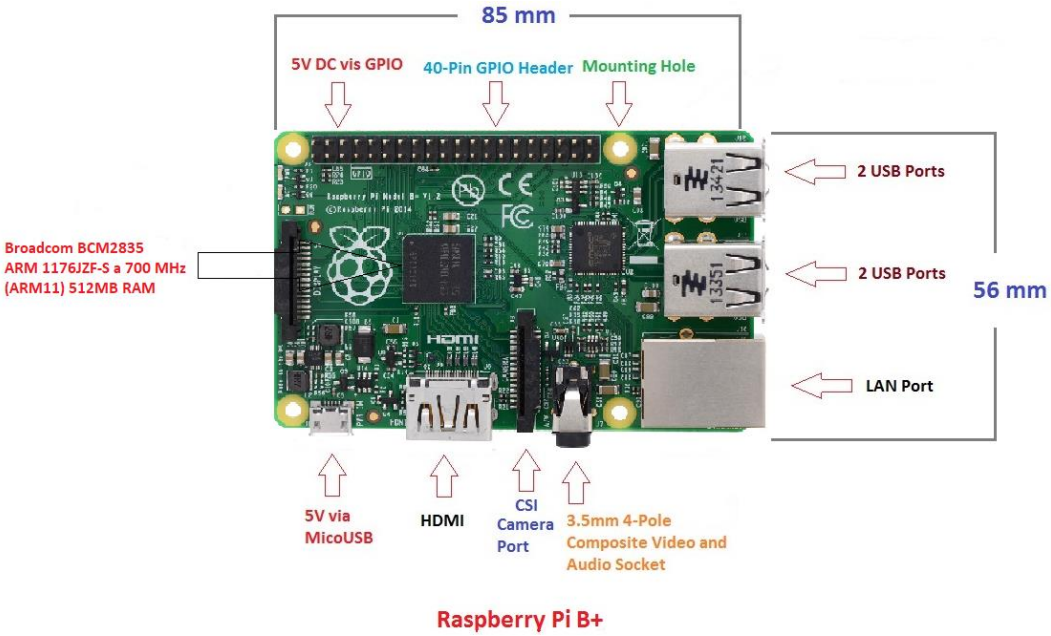


Fig. 2.1. Raspberry Pi B+[3], [4]

- **Raspberry Pi 3B+**: es una actualización del modelo Raspberry Pi 3B. El modelo anterior abre una nueva era en la Raspberry Pi con la inclusión de comunicaciones inalámbricas, como Bluetooth y Wifi. Entre las mejoras de este modelo destacan una mayor velocidad en el procesador, mayor velocidad en el puerto Ethernet, compatibilidad con la banda Wifi de 5GHz y una optimización de la potencia consumida.

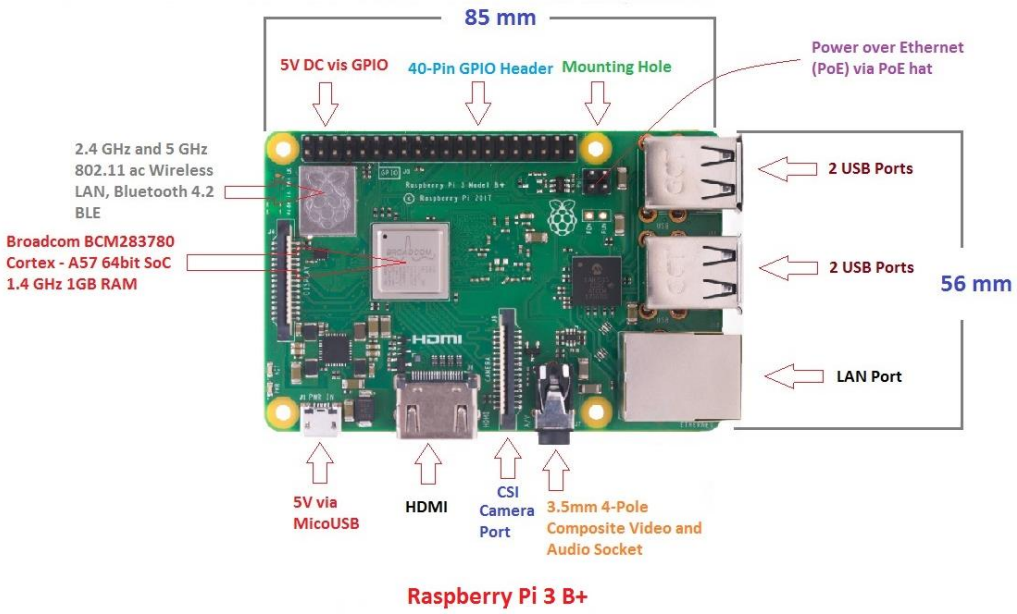


Fig. 2.2. Raspberry Pi 3B+[3]

2.2 Alimentación

Como se ha mencionado en apartados anteriores, se van a utilizar dos modelos de Raspberry Pi diferentes: el B+ y el 3B+. Para hacer funcionar ambos modelos es necesario conectar una fuente de alimentación o un transformador. La Raspberry Pi se puede

alimentar de dos formas: a través del puerto micro USB (recomendada) y a través de los pines GPIO de 5V. La diferencia entre un modo u otro es que la alimentación a través del puerto micro USB proporciona más seguridad ya que contiene a su entrada reguladores y fusibles que protegen a la placa ante sobretensiones y picos de corriente. Sin embargo, los pines GPIO de 5V no poseen ningún tipo de circuito de protección a su entrada por lo que se debe tener especial cuidado en la conexión para no dañar la Raspberry Pi permanentemente.

En relación con lo anterior, es necesario cumplir con un mínimo de potencia en su alimentación, es decir, proporcionar una tensión y una corriente adecuada. La tensión mínima soportada es de 4.65V y la máxima 5.2V. Superar esta última tensión puede quemar la Raspberry Pi. Por el contrario, no llegar a esta o no proporcionar los vatios suficientes producirá su apagado.[1]

Según la web oficial de la Raspberry Pi el transformador que incluye la Raspberry Pi es capaz de proporcionar una tensión de 5.1V junto con una corriente máxima de 2A. Sin embargo, el modelo B+ y el modelo 3B+ tienen distintos consumos.[1]

Se ha realizado una tabla con el valor de consumo máximo desglosado por el gasto producido por los puertos USB y por el total consumido de la placa donde se encuentra el procesador, la RAM, los pines GPIO, etc. Asimismo, se incluye una columna con la corriente mínima recomendada que debe proporcionar el trasformador o la fuente de alimentación que se utilice. Los valores se muestran en la siguiente tabla.

TABLA 2.1. CONSUMO APROXIMADO DE LA RASPBERRY PI[1]

Modelo	Fuente de alimentación recomendada	Máxima corriente consumida por todos los puertos USB	Máxima corriente consumida por la placa
B+	1.8A	600mA/1.2A	330mA
3B+	2.5A	Limitado por la fuente de alimentación, la placa y los conectores.	150 mA

Por otra parte, se incluye otra medida de los consumos sacada de la documentación de la Raspberry Pi, pero en este caso con los consumos en distintas situaciones. Los resultados se muestran en la siguiente tabla.

TABLA 2.2. PRUEBAS DE CONSUMO EN DISTINTAS SITUACIONES EN LA RASPBERRY PI[2]

		B+ (A)	3B+ (A)
Boot	**Max**	0.26	0.75
	Avg	0.22	0.35
Idle	**Avg**	0.2	0.3
	Max	0.3	0.55
Reproducción de video	**Avg**	0.22	0.33
	Max	0.35	1.34
Máximo rendimiento	**Avg**	0.32	0.85

	Max	0.26	0.75

Las condiciones de las medidas realizadas en la tabla anterior se han realizado bajo la misma imagen Raspbian para ambos modelos, con un monitor conectado por HDMI, un teclado y un ratón USB, y sin contar la corriente consumida en los puertos USB. Además, en el caso del modelo 3B+ fue conectado a una red WIFI.

En conclusión, para alimentar el modelo B+ es necesario como mínimo una fuente de alimentación de 5V y 1.8A, es decir, una potencia mínima de 9W. Por el contrario, para el modelo 3B+ se necesita como mínimo una fuente de alimentación de 5V y 2.5A, una potencia mínima 12.5W. Se debe tener en cuenta que las potencias están calculadas contando con que la Raspberry Pi está a máximo rendimiento y con los puertos USB consumiendo al máximo, aunque escasas veces sobrepasaremos a estos límites siempre se ha de tener cierto margen de seguridad.

2.3 Configuración inicial

Se han de seguir una serie de pasos para poder trabajar de la manera requerida con la Raspberry Pi.

El primer paso es la instalación de un sistema operativo que permita interactuar con los distintos componentes que componen su hardware. Para ello, se debe elegir el sistema operativo para las funciones que necesitemos desempeñar.

Por otra parte, una vez se haya iniciado el sistema operativo, es necesario realizar una serie de configuraciones para poder comunicarse con la Raspberry Pi a través de un ordenador, y para activar ciertas funciones que se necesitaran más adelante.

2.3.1 Elección de SO

Para hacer funcionar la Raspberry Pi es necesario instalar el sistema operativo. Para ello, debemos disponer de una tarjeta microSD con un mínimo de 8GB o 4GB de almacenamiento dependiendo del sistema operativo que se vaya a utilizar.

De acuerdo con los requisitos del proyecto se busca un sistema operativo que proporcione las funcionalidades necesarias para permitir el uso de las Raspberry Pi como un PC.

La distribución que se ha elegido es Raspbian Debian Jessie, por dos motivos principales. El primero es que no es necesario la instalación del software que se utiliza para el control de las interfaces del set de pines de la Raspberry Pi. A esto se le añade, la inclusión de las librerías necesarias para el desarrollo en C. Además, posee detrás una gran comunidad en Internet lo que supone mayor facilidad para la búsqueda de contenido y para la solución de errores. De igual forma, es retrocompatible con los dos modelos de Raspberry Pi que se van a utilizar en el proyecto.

Para su descarga, se accede a su página web oficial, donde se obtiene la distribución en formato ISO.

2.3.2 Instalación del SO

Seleccionado el sistema operativo que se va a utilizar. El siguiente paso a seguir es el proceso de instalación de la distribución en formato ISO en un soporte físico, este es una tarjeta microSD.

Con referencia a la instalación, se utiliza la herramienta balenaEtcher que es la recomendada en la documentación de la Raspberry Pi.

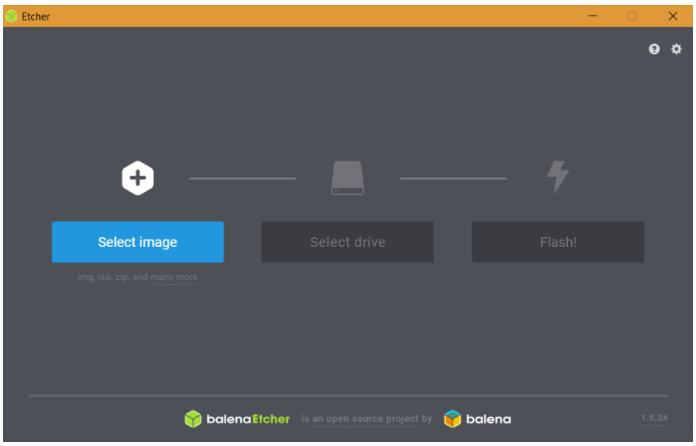


Fig. 2.3. Pantalla principal de balenaEtcher[5]

Finalmente, una vez termina la instalación ya está la microSD preparada.

2.3.3 Configuración acceso remoto

Al terminar con la instalación de Raspbian, a continuación se llevan a cabo una serie de configuraciones.

Atendiendo a los requisitos del proyecto, es necesario acceder de manera remota a la Raspberry Pi a través de un cable que conecta el puerto Ethernet de la Raspberry Pi con el puerto Ethernet del ordenador.

Para llevar a cabo la comunicación por acceso remoto se emplea el protocolo SSH. Para poder establecer la conexión las interfaces Ethernet del ordenador y de la Raspberry Pi deben estar dentro del mismo rango de direcciones IP. Sin embargo, en ambos casos está por defecto activado el protocolo DHCP esto supone que la dirección IP y la máscara subred se asigna automáticamente. En consecuencia, cada vez que alguno de los dos se reinicia se le asigna otra dirección IP distinta a la interfaz Ethernet a la que tenía antes de ser reiniciado, por lo que nunca compartirán direcciones IP que se encuentren en el mismo rango que marca la máscara subred de cada uno. En definitiva, para la comunicación a través de SSH se han de configurar una serie parámetros y características tanto en la Raspberry Pi como en el ordenador.

El primer paso es establecer una dirección IP estática para poder acceder a la Raspberry Pi. Abrimos el Panel de Control y entramos al apartado Red e Internet. En este apartado accedemos a la “Configuración avanzada del adaptador”. Después damos clic derecho a la interfaz Ethernet y accedemos a Propiedades. En la categoría “Protocolo de Internet versión 4 (TCP/IPv4)” aparece la ventana de configuración que buscábamos.

Marcamos la opción “Usar la siguiente dirección IP” e introducimos la dirección IP en la que nos hemos fijado anteriormente que en este caso es *.*.*.* y la máscara subred predeterminada 255.255.255.0. La configuración se muestra en la siguiente figura.

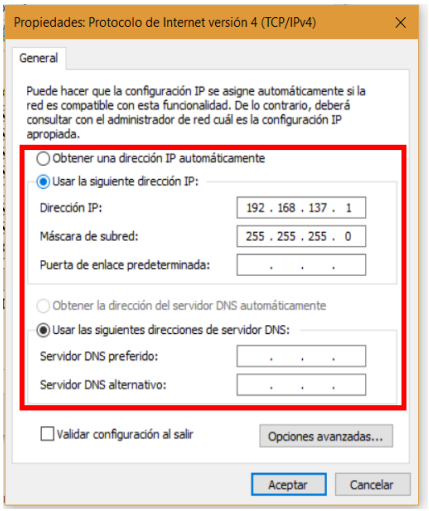


Fig. 2.4. Configuración para SSH en el ordenador

Por otra parte, se necesita una aplicación cliente que sea capaz de conectarse a través del protocolo SSH. Para este cometido utilizamos la herramienta PuTTY.

Ejecutamos PuTTY y nos aparece la ventana que se muestra en la siguiente figura.

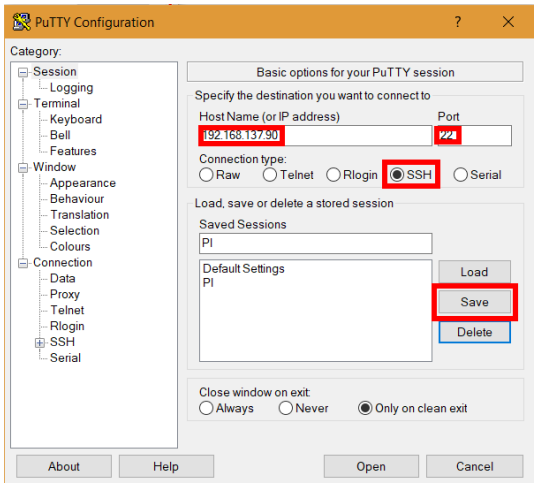


Fig. 2.5. PuTTY

La configuración de la sesión se configura de la siguiente manera:

- **Host name (or IP address)**. En este apartado se introduce la dirección IP estática que se corresponde con la Raspberry Pi
- **Port**. Es el puerto asociado a la conexión SSH.
- **Connection type**. Tipo de protocolo a utilizar en la conexión.

Todas estas configuraciones las guardamos en un perfil llamado ‘PI’ mediante el botón “Save”, así no será necesario introducir de nuevo la configuración cada vez que cerremos la aplicación.

En lo que concierne a la configuración del ordenador ya estaría terminada.

En cuanto a la Raspberry Pi, es necesario activar el protocolo SSH que viene desactivado por defecto. Existen dos maneras de activar el protocolo SSH. La primera requiere de encender la Raspberry Pi con la microSD insertada, y conectar una pantalla a través del HDMI, un teclado y un ratón USB. De esta manera podremos activar a través del menú de configuración de la Raspberry Pi manualmente el protocolo. Sin embargo, el gran inconveniente supone tener que disponer todos estos periféricos y que sean compatibles con la Raspberry Pi. Asimismo, ya que queremos manejar la Raspberry Pi de manera remota, lo suyo es acceder sin tener que hacer ninguna manipulación física directa extra más allá de la conexión que hay entre el ordenador y esta.

La segunda manera consiste en crear un fichero vacío con un editor de texto cualquiera. Es muy importante que este archivo no contenga ningún tipo de texto. Finalmente, una vez creado se renombra a SSH y se copia a la carpeta raíz de la microSD como se muestra en la siguiente figura.

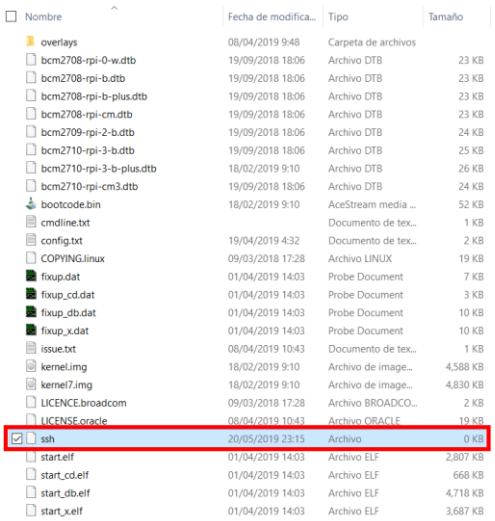


Fig. 2.6. Raíz de la microSD

De esta forma, el protocolo SSH quedaría permanentemente activado.

Por otro lado, para conectarse a través del protocolo SSH se necesita conocer la dirección IP asociada a la interfaz (puerto) Ethernet de la Raspberry Pi. Además, está dirección IP debe entrar dentro del rango de la dirección establecida en la interfaz Ethernet del ordenador. Este rango parte de X.Y.Z.0 hasta X.Y.Z.255, siendo X, Y, Z los primeros 3 números de la dirección IP del ordenador. Por supuesto, en este rango se excluye la dirección IP de este último.

En este caso, al ser la dirección IP del ordenador 192.168.137.1, el rango de direcciones IP que podría establecerse en la Raspberry Pi es del 192.168.137.0 a 192.168.137.255 a excepción de la dirección IP del ordenador. En consecuencia, se escoge una dirección IP cualquiera para la Raspberry Pi, en este caso es 192.168.137.90 la elegida.

Por otro parte, para establecer esta dirección IP a la interfaz Ethernet de la Raspberry Pi es necesario editar el archivo 'cmdline.txt' que se encuentra en la raíz de la tarjeta microSD. En este archivo es necesario añadir la siguiente línea: 'ip=192.168.137.90'.

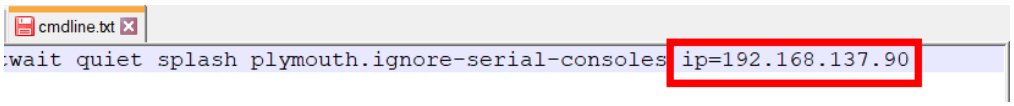


Fig. 2.7. Configuración de IP fija en la Raspberry Pi

Una vez editado el archivo, se inserta la microSD en la Raspberry Pi y se conecta el transformador para encenderla. Durante un tiempo aproximado de 30s la Raspberry Pi carga las distintas capas y configuraciones del sistema operativo.

Si no hay ningún problema, el puerto Ethernet de la Raspberry Pi con un led amarillo parpadea indicando que la conexión física funciona sin problema.

Para comunicarse con la Raspberry Pi, abrimos la aplicación PuTTY y cargamos el perfil 'Pi'. Después, le damos a 'Open' para iniciar la conexión. Si todo ha ido bien aparece la pantalla de login de la Raspberry Pi cuyo usuario y contraseña es, 'pi' y 'raspberry'. Sin embargo, si la conexión no es exitosa tras un tiempo de espera (timeout) el programa lanza el error de la siguiente figura

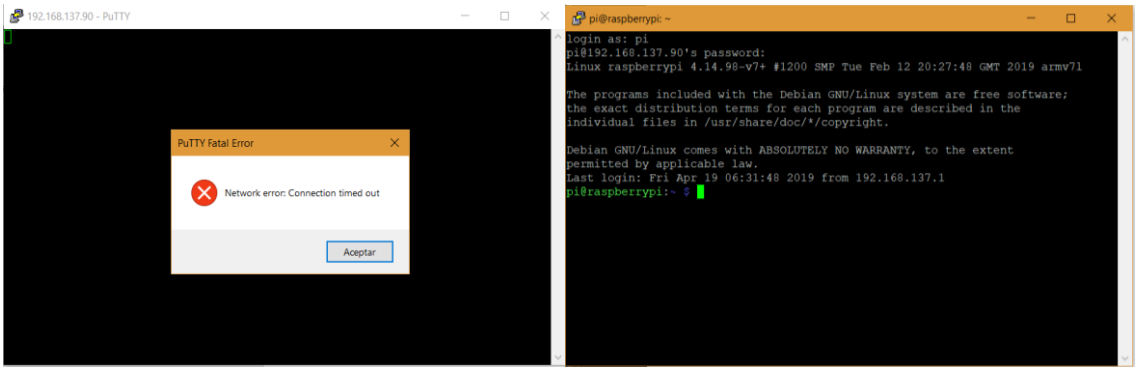


Fig. 2.8. Conexión SSH en PuTTY

2.3.4 Configuración Raspberry Pi

Una vez establecida la conexión la Raspberry Pi, se procede a la configuración dentro del sistema operativo Raspbian para la activación de los protocolos de comunicación serial fijados en los requisitos. Las comunicaciones serial, por defecto desactivadas, que se van a utilizar son: SPI, I2C y UART.

Para la activación de los protocolos mencionados, es necesario acceder al menú de configuración para la Raspberry Pi que incorpora Raspbian. Para ello, desde la línea de comandos introducimos 'sudo raspi-config', y nos aparece la siguiente figura.

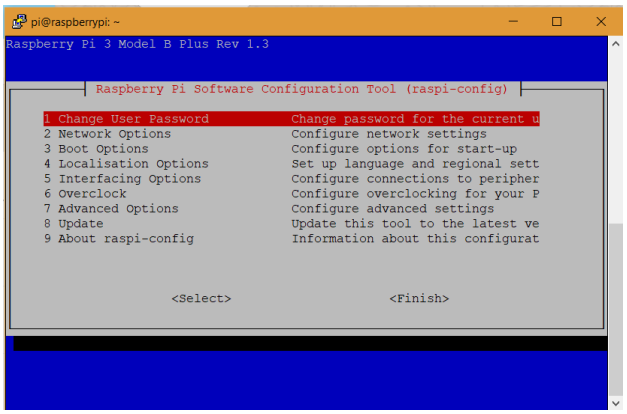


Fig. 2.9. Menú de configuración Raspberry Pi

Con las flechas del teclado, navegamos hasta la opción ‘5 Interfacing Options’ y pulsamos ‘Enter’. Entra las distintas interfaces que aparecen, seleccionamos las siguientes ‘P4 SPI’ para activar el protocolo SPI, ‘P5 I2C’ para activar el protocolo I2C y ‘P6 Serial’ para activar el protocolo UART. Cada vez que se selecciona una opción, aparece una ventana para confirmar la activación.

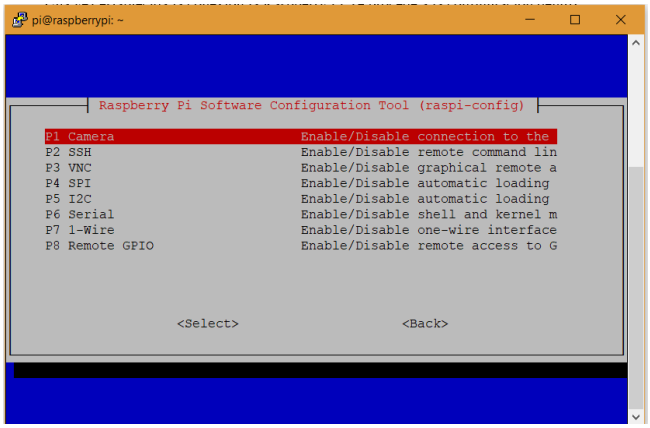


Fig. 2.10. 'Interfacing Options' menú de configuración Raspberry Pi

Finalmente, terminada la activación de los protocolos serial, navegamos hasta ‘Finish’ para cerrar el menú de configuración.

Todos los protocolos activados a través de este menú de configuración permanecen activos, aunque esta se reinicie o se apague la Raspberry Pi.

2.4 GPIO

Una de las características más destacada de las Raspberry Pi es el set conexiones que incluye en forma de pines. Es utilizado por multitud de proyectos de desarrollo dado la versatilidad que ofrece debido a que se pueden configurar indistintamente como entradas o salidas.

Normalmente la mayor parte de modelos poseen 40 pines GPIO. Sin embargo, existen otros modelos con menor número de pines, aunque se disponen en el mismo orden,

haciéndolos compatibles entre distintos modelos. En este caso ambos modelos utilizados poseen 40 pines GPIO, la disposición de estos pines se refleja en la siguiente figura.

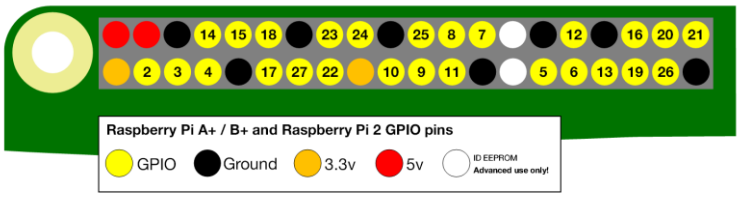


Fig. 2.11. Pines GPIO de la Raspberry Pi[6]

Los pines de GPIO están destinados a diferentes tareas y, según este último criterio se agrupan en varios tipos[7]:

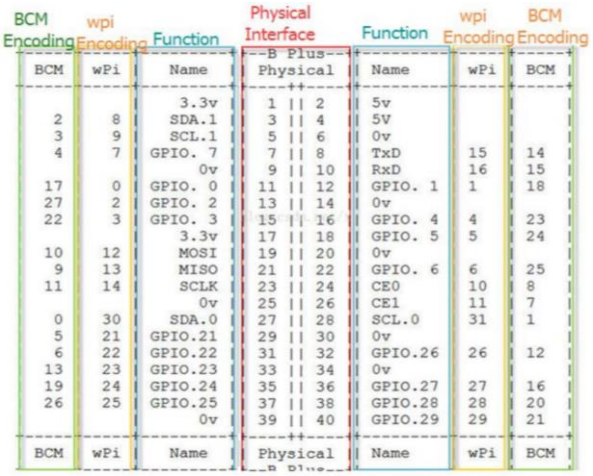
- **Comunes**. Estos pines son aquellos que se pueden configurar como entradas o salidas. Soportan una tensión máxima de 3.3V, que representa el nivel alto, '1', y el nivel bajo lo representa la tensión nula, '0'. Dentro de este set de pines existe otra funcionalidad muy interesante como el PWM.
- **Alimentación**. Existen tres pines que proporcionan distintos niveles de voltaje, dos de 5V y uno de 3.3V. Los dos pines de 5V proporcionan dicha salida de tensión si se conecta la Raspberry Pi a una fuente de tensión a través del puerto micro USB. A su vez a través de estos pines se puede alimentar la Raspberry Pi sin necesidad de utilizar un cable micro USB. Sin embargo, el pin de 3.3V solo tiene la función de ofrecer dicha tensión como salida, no se puede alimentar a la Raspberry Pi a través de él. Por otra parte, también existen 6 pines conectados su masa.
- **Comunicaciones serial**. Existen varios pines especiales están reservados para los protocolos de comunicación serial.
    - **SPI**. Para este protocolo están destinados 11 pines dividido en dos componentes:
        - **SPI 1**. CE1, CE0, CE2, MISO, MOSI y SCLK,
        - **SPI 0**. CE0, CE1, MOSI, MISO y SCLK.
    - **I2C**. Para el funcionamiento de este protocolo están reservados 4 pines: DATA, CLOCK, EEPROM DATA y EEPROM CLOCK.
    - **UART**. En este protocolo se utilizan 2 pines: RX y TX.

Aparte, también existen distintos criterios en la forma de numeración que se asigna a cada uno de los pines. Los criterios son los siguientes[8]:

- **Criterio GPIO**. De acuerdo con este modo, los pines se numeran de la manera en que se disponen físicamente en la placa de la Raspberry Pi. El pin numero 1 es el

que se posiciona más arriba a la izquierda teniendo la Raspberry Pi dispuesta con los puertos USB hacia arriba.

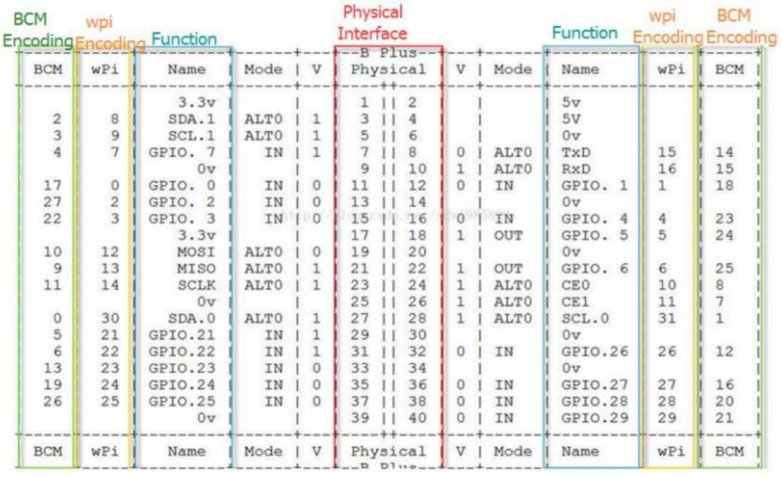
- **Criterio BCM**. En este criterio se numeran los pines de acuerdo con la numeración tienen asignados los puertos dispuestos en el procesador Broadcom de la Raspberry Pi.
- **Criterio Wiring Pi**. Sigue la misma disposición que el criterio GPIO.



BCM	wPi	Name	Physical	Name	wPi	BCM
		3.3v	1 || 2	5v		
2	8	SDA.1	3 || 4	5V		
3	9	SCL.1	5 || 6	0v		
4	7	GPIO. 7	7 || 8	TxD	15	14
		0v	9 || 10	RxD	16	15
17	0	GPIO. 0	11 || 12	GPIO. 1	1	18
27	2	GPIO. 2	13 || 14	0v		
22	3	GPIO. 3	15 || 16	GPIO. 4	4	23
		3.3v	17 || 18	GPIO. 5	5	24
10	12	MOSI	19 || 20	0v		
9	13	MISO	21 || 22	GPIO. 6	6	25
11	14	SCLK	23 || 24	CE0	10	8
		0v	25 || 26	CE1	11	7
0	30	SDA.0	27 || 28	SCL.0	31	1
5	21	GPIO.21	29 || 30	0v		
6	22	GPIO.22	31 || 32	GPIO.26	26	12
13	23	GPIO.23	33 || 34	0v		
19	24	GPIO.24	35 || 36	GPIO.27	27	16
26	25	GPIO.25	37 || 38	GPIO.28	28	20
		0v	39 || 40	GPIO.29	29	21
BCM	wPi	Name	Physical	Name	wPi	BCM

Fig. 2.12. Pines GPIO con sus modos predeterminados en los tres criterios (Amarillo: wPi, Verde: BCM, Azul: GPIO)

Al iniciar la Raspberry Pi, el set de pines se inicializa con un valor predeterminado que viene asignado por el procesador Broadcom, tanto su modo (entrada o salida) como su valor. Es muy importante tener en cuenta este detalle a la hora de conectar componentes a las interfaces GPIO de la Raspberry Pi. Los valores predeterminados del chip Broadcom se muestra en la figura.



BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM
		3.3v			1 || 2			5v		
2	8	SDA.1	ALT0	1	3 || 4			5V		
3	9	SCL.1	ALT0	1	5 || 6			0v		
4	7	GPIO. 7	IN	1	7 || 8	0	ALT0	TxD	15	14
		0v			9 || 10	1	ALT0	RxD	16	15
17	0	GPIO. 0	IN	0	11 || 12	0	IN	GPIO. 1	1	18
27	2	GPIO. 2	IN	0	13 || 14			0v		
22	3	GPIO. 3	IN	0	15 || 16	0	IN	GPIO. 4	4	23
		3.3v			17 || 18	1	OUT	GPIO. 5	5	24
10	12	MOSI	ALT0	0	19 || 20			0v		
9	13	MISO	ALT0	1	21 || 22	1	OUT	GPIO. 6	6	25
11	14	SCLK	ALT0	1	23 || 24	1	ALT0	CE0	10	8
		0v			25 || 26	1	ALT0	CE1	11	7
0	30	SDA.0	ALT0	1	27 || 28	1	ALT0	SCL.0	31	1
5	21	GPIO.21	IN	1	29 || 30			0v		
6	22	GPIO.22	IN	1	31 || 32	0	IN	GPIO.26	26	12
13	23	GPIO.23	IN	0	33 || 34			0v		
19	24	GPIO.24	IN	0	35 || 36	0	IN	GPIO.27	27	16
26	25	GPIO.25	IN	0	37 || 38	0	IN	GPIO.28	28	20
		0v			39 || 40	0	IN	GPIO.29	29	21
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM

Fig. 2.13. Pines GPIO con sus valores y modos predeterminados en los tres criterios (Amarillo: wPi, Verde: BCM, Azul: GPIO)

Se debe tener en cuenta antes de realizar una conexión con los pines GPIO, comprobar la tensión y la corriente que se va a requerir.

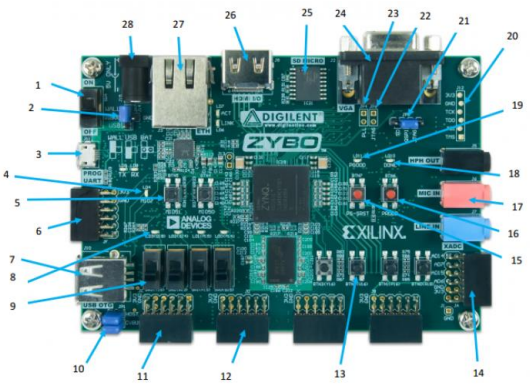
Los pines GPIO son capaces de conducir 50mA a 3.3V distribuido a lo largo de todos ellos. Normalmente el valor nominal que conduce un pin GPIO es de 3 mA, y en el mejor de los casos es capaz de soportar un máximo de 16 mA de manera segura.

Por otro lado, se deben tomar precauciones a la hora de conectar componentes a este set de conexiones, ya que no tiene buffers de protección. Esto supone que una conexión errónea puede dañar la Raspberry Pi de forma permanente.

3 ZYBO ZYNQ-7000

3.1 Introducción

"ZYBO (Zynq Board) es una plataforma de desarrollo de software integrado y software de nivel básico, lista para usar, creada alrededor del miembro más pequeño de la familia Xilinx Zynq-7000, el Z-7010. El Z-7010 se basa en la arquitectura Xilinx All Programmable System-on-Chip (SoC AP), que integra estrechamente un procesador ARM Cortex-A9 de doble núcleo con la lógica de Array de Puertas Programable de Campo (FPGA) de la serie Xilinx 7. Cuando se combina con el amplio conjunto de periféricos multimedia y de conectividad disponibles en el ZYBO, el Zynq Z-7010 puede albergar un diseño de sistema completo. Las memorias integradas, la E/S de video y audio, la ranura USB de doble función, Ethernet y SD tendrán su diseño listo y listo, sin necesidad de hardware adicional. Además, hay seis puertos Pmod disponibles para colocar cualquier diseño en una ruta de fácil crecimiento."[9]



Callout	Component Description	Callout	Component Description
1	Power Switch	15	Processor Reset Pushbutton
2	Power Select Jumper and battery header	16	Logic configuration reset Pushbutton
3	Shared UART/JTAG USB port	17	Audio Codec Connectors
4	MIO LED	18	Logic Configuration Done LED
5	MIO Pushbuttons (2)	19	Board Power Good LED
6	MIO Pmod	20	JTAG Port for optional external cable
7	USB OTG Connectors	21	Programming Mode Jumper
8	Logic LEDs (4)	22	Independent JTAG Mode Enable Jumper
9	Logic Slide switches (4)	23	PLL Bypass Jumper
10	USB OTG Host/Device Select Jumpers	24	VGA connector
11	Standard Pmod	25	microSD connector (Reverse side)
12	High-speed Pmods (3)	26	HDMI Sink/Source Connector
13	Logic Pushbuttons (4)	27	Ethernet RJ45 Connector
14	XADC Pmod	28	Power Jack

Fig. 3.1. Diagrama de Zybo[10]

3.2 Alimentación

La Zybo tiene diferentes formas de conectar la alimentación: a través del puerto UART/JTAG USB, el puerto Jack o una fuente de tensión externa.[10]

Dependiendo de la forma de alimentación escogida se debe establecer el 'power jumper' que posee la Zybo en una de sus tres posiciones disponibles: USB, 'Wall' con el Jack y batería externa.[10]

Fig. 3.2. Posiciones 'power jumper'[10]

Si la conexión de la alimentación se hace través de puerto UART/JTAG USB, según las especificaciones un puerto USB 2.0 puede proporcionar un máximo de 0.5A a 5V. Únicamente la Zybo en reposo consume 0.2A, y para diseños no muy complicado una media de 0.38A. Por lo tanto, si necesitamos conectar componentes externos a través del HDMI, VGA o USB no dispondríamos de potencia suficiente, pero este no es nuestro caso. Esta forma de alimentar la Zybo se utiliza posteriormente a la hora de realizar las pruebas de comunicación USB-UART con la Raspberry Pi.[10]

Si la conexión de la alimentación se establece a través del 'power jack' la fuente debe conectarse a través de un conector coaxial y proporcionar una tensión de entre 4.5V y 5.5V (12.5W como mínimo). Esta forma de alimentar la Zybo se utiliza posteriormente a la hora de las pruebas con la alimentación que le proporciona el sistema de control.[10]

Si la conexión se produce a través de una fuente de tensión externa conectando el cable positivo al pin del centro JP7 y el cable de masa al pin J14. El voltaje máximo que soporta la Zybo sin que sufra ningún daño es de 5.5V, y el mínimo de 4.6V.[10]

3.3 Puertos Pmod

La Zybo contiene una serie de puertos de entrada y salida denominados 'Pmod'. Estos puertos son de 2x6 y proporcionan una tensión de 3.3V, tienen implementado un sistema de protección que regula la corriente para evitar que se produzcan daños.[10]

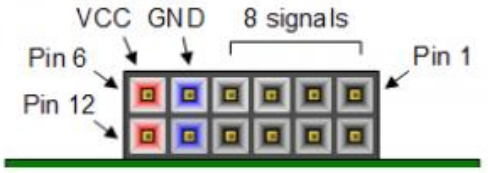


Fig. 3.3. Diagrama Pmod[10]

Los pines VCC y GND son capaces de proporcionar una corriente máxima de 1A. La Zybo implementa cinco de estos tipos de puerto. Más adelante, se utilizarán sus pines como salidas y/o entradas para las interfaces de comunicación que se implementarán en ambos extremos de la comunicación (Zybo y sistema de control).[10]

Cabe añadir, que a través de estos pines son compatibles para externalizar una señal de reloj, por defecto la máxima frecuencia es de 50MHz.[10]

Por otro lado, la Zybo soporta señales externas de ‘reset’ que permite reiniciar el chip en su totalidad. Además, implementa dos botones para reiniciar el chip, pero con diferentes comportamientos. El botón PROG reinicia todo el chip y el botón PS-SRST que reinicia el chip sin reiniciar la rutina de depuración.[10]

4 DISEÑO HARDWARE DEL SISTEMA DE CONTROL

4.1 Introducción

El sistema de control es el núcleo central del proyecto en él se aglutinan la mayoría de las funciones de control de los dispositivos, alimentación y conexiones. En la *Fig. 4.1* se presenta el esquema completo que se va a llevar a cabo para realizar el experimento.

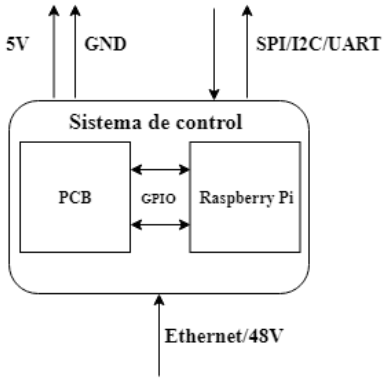


Fig. 4.1. Esquema del Sistema Control

En el sistema de control podemos distinguir dos bloques. La PCB que contiene el hardware necesario para conectar los dispositivos (Zybo), su alimentación y su control, el cuál deberemos diseñar y fabricar, y la Raspberry Pi que posee el software necesario para hacer funcionar el sistema de control y los protocolos de comunicación. Ambos bloques están conectados entre sí mediante los pines GPIO de la Raspberry Pi a través de un cable de 40 pines en un conector de 2x20 pines.

La placa (PCB) está compuesta por una serie de componentes que proporcionan la alimentación adecuada a los dispositivos y que permiten controlar su apagado y encendido. Además, contiene las conexiones necesarias tanto para la conexión del dispositivo como para la comunicación mediante los distintos protocolos que se establecen en los requisitos.

El otro bloque, la Raspberry Pi, ejecuta en lenguaje C los programas con las interfaces de comunicación necesarias para el control de los dispositivos y el intercambio de información.

La Raspberry Pi está en constante comunicación con un ordenador a través de un puerto Ethernet. Desde el ordenador se desarrolla, compila y elige qué programa a ejecutar en la Raspberry Pi.

Los dispositivos conectados al sistema de control implementan el otro extremo de la interfaz de comunicación de los protocolos que se definen en los requisitos. Permitiendo la comunicación entre estos y el sistema de control a través de la Raspberry Pi.

4.2 Diseño del hardware

Como se ha mencionado anteriormente, el sistema de control se divide en dos. En este apartado nos encargaremos del diseño del bloque de la PCB que es la placa que integra

los componentes que permiten la conexión y la alimentación del sistema, entre otras funciones.

En primer lugar, se dividen los requisitos en bloques funcionales y en cada uno se lleva a cabo un análisis de los componentes necesarios para su funcionamiento, la elección de manera justificada del modelo y su implementación en un circuito.

En segundo lugar, se ejecutan una serie de simulaciones virtuales del circuito antes de implementarlo en una placa física.

En tercer lugar, se procede al diseño del circuito en PCB donde se describe las dimensiones de la placa, la ubicación de los componentes, el diseño de las pistas, y posteriormente su fabricación.

Por último, se llevan a cabo una serie de pruebas funcionales para verificar que el diseño de la placa coincide con el del circuito y que se ha fabricado sin ningún tipo de error.

4.2.1 Herramientas utilizadas

1. Multisim. Consultar Anexo J.
2. PSpice Capture CIS. Consultar Anexo K.
3. OrCAD Layout Plus. Consultar Anexo K.

4.2.2 Componentes asociados de acuerdo con los requisitos

En la introducción del apartado se menciona que, dentro del sistema de control en la parte de la placa, se distinguen varios bloques funcionales: el control del dispositivo, conexión del dispositivo y la Raspberry PI., comunicación con el dispositivo, y la conexión y conversión de tensión.

Con ayuda de la herramienta OrCAD Capture CIS a medida que se constituyen los distintos componentes y conexiones, se van implementando los distintos bloques descritos en un circuito esquemático

4.2.2.1 Bloque de alimentación y conversión

Uno de los requisitos principales a cumplir es alimentar a los dispositivos, en este caso, las Zybo y la Raspberry Pi que se conectan al sistema de control. Según las especificaciones establecidas el rango de funcionamiento de la alimentación que se ha de implementar, deber ser de una tensión de 5V para que las Zybo y la Raspberry Pi funcionen, y no sufran daños.

El principal problema que nos encontramos es que la fuente de alimentación de la que se dispone proporciona una salida de 48V, lo que supondría quemar nuestro circuito junto con las Zybo y la Raspberry Pi. En consecuencia, para evitarlo se decide implementar un componente denominado convertidor de corriente continua, que es capaz de convertir 48V en 5V. Para saber más información de este componente consultar el Anexo B.

No sólo es necesario que tenga una tensión adecuada, sino que sea capaz de proporcionar la potencia suficiente como para hacer funcionar los dispositivos mencionados de manera simultánea. Para este cometido, se ha establecido una potencia máxima de 5W por

dispositivo, suficiente para hacer compatible nuestro sistema de control con cualquier tipo de FPGA estándar en el mercado. El objetivo de esto es hacer el sistema de control compatible con una mayor cantidad de dispositivos.

Se incluyen dos convertidores, uno necesario para alimentar a las Zybo, y otro para alimentar a la Raspberry Pi y al circuito diseñado en la placa. El principal motivo por el que se incluyen dos en vez de uno, es para evitar que la Raspberry Pi pierda la alimentación si algún dispositivo falla o por si los dispositivos utilizados no comparten la misma tensión que esta, lo que permite una mayor compatibilidad con los dispositivos disponibles en el mercado.

El convertidor encargado de alimentar las Zybo debe convertir la tensión de 48V a 5V. Como se ha mencionado anteriormente, se ha establecido que la potencia por dispositivo soportado por el sistema sea de 5W, de acuerdo con los cálculos:

$$Potencia\ requerida = 5\ W \times 4(dispositivos) = 20\ W \quad (4.1)$$

Esto significa que a una tensión de 5V la corriente mínima requerida debe ser de 4A en total (1A por dispositivo).

El segundo convertidor, se encarga de alimentar a la Raspberry Pi y al resto de componentes del circuito. Como el anterior, convierte la tensión de entrada de 48V a 5V en su salida.

En el apartado de la Raspberry Pi, se muestra que al consumo máximo del modelo B+ es de 9W, mientras que el del modelo 3B+ es de 12.5W. Por lo tanto, nos fijamos en este último modelo para establecer la potencia mínima que debe proporcionar el convertidor. A esto se le debe añadir, el consumo del resto de componentes que se han de implementar.

Se calcula que al ser componentes de bajo consumo es suficiente con añadir un margen de seguridad de 5W.

$$P_{min} = Consumo\ máximo\ (3B+) + Consumo\ máximo\ (componentes) \quad (4.2)$$

$$P_{min} = 12.5W + 5W \cong 18W \quad (4.3)$$

Según los cálculos realizados el convertidor debe proporcionar una potencia mínima de aproximadamente 18W. En consecuencia, a una tensión de 5V debe ser capaz de proveer una corriente máxima de al menos 3.6A.

Como indicadores para el funcionamiento de la alimentación, se conecta un led paralelo a la salida de cada convertidor para facilitar el reconocimiento rápido de algún problema en esta parte del circuito. Si el led está encendido significa que el convertidor proporciona la tensión adecuada en su salida y si está apagado, puede significar dos cosas. La primera que el convertidor esté apagado de forma voluntaria y la segunda que haya algún problema en la alimentación del sistema.

Otra característica que se describe en los requisitos es que los convertidores lleven incorporado un puerto de control remoto para controlar su apagado y encendido

eléctricamente. A este puerto se conecta un pin GPIO de la Raspberry Pi que se encarga su control. Si se configura el pin GPO a nivel bajo mantiene el convertidor encendido y si se establece a nivel alto, este se apaga.

En cuanto a las características eléctricas asociadas al convertidor, se busca un modelo que contenga protección ante cortocircuito, por sobretensión, por exceso de temperatura y por sobrecarga. La importancia de estas características es que nos proporciona una mayor garantía para proteger el convertidor y el resto de los componentes del circuito.

En cuanto a las características físicas del convertidor, se busca implementar uno cuyo tamaño sea contenido y posea una buena disipación del calor durante su funcionamiento. Asimismo, al ser un componente con un precio alto, se tienen en cuenta los costes asociados a la elección del tipo de convertidor. El modelo que se busca es aquel que tenga un precio contenido en relación con las características que se necesitan.

De entre todos los modelos, se escoge el modelo de convertidor Meanwell SKMW30G-05 (Consultar Anexo B) debido a que cumple con todas las características necesarias y además por un precio menor que el resto de los convertidores del mercado. Sin embargo, consultando las hojas de características este convertidor tiene una desventaja debido a la forma de controlar su apagado y encendido.

REMOTE CONTROL	Power ON: R.C. ~ -Vin >3.5~75Vdc or open circuit ; Power OFF: R.C. ~ -Vin <1.2Vdc or short

Fig. 4.2. Control remoto del convertidor (Hoja de características)

Según su hoja de características, este modelo tiene dos formas para poder apagarlo y encenderlo. Una de las maneras para el control remoto es:

- Apagado. Se cortocircuita los pines R.C y $-V_{in}$.
- Encendido. Se deja circuito abierto entre los pines R.C y $-V_{in}$.

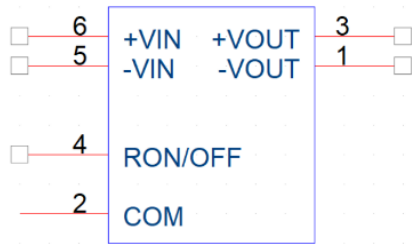


Fig. 4.3. Convertidor apagado (Método 1)

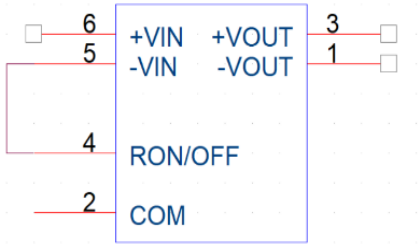


Fig. 4.4. Convertidor encendido (Método 1)

La otra manera de controlar su apagado y encendido es:

- Apagado. Se aplica una diferencia de voltaje menor que 1,2V entre R.C y $-V_{in}$ para el apagado.
- Encendido. Se aplica una diferencia de voltaje entre 3,5V~75V entre R.C y $-V_{in}$ para su encendido.

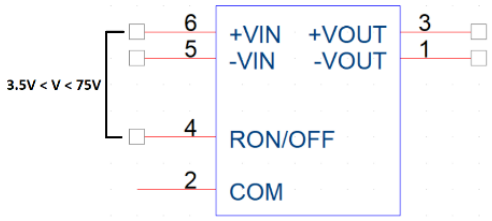


Fig. 4.5. Convertidor apagado (Método 2)

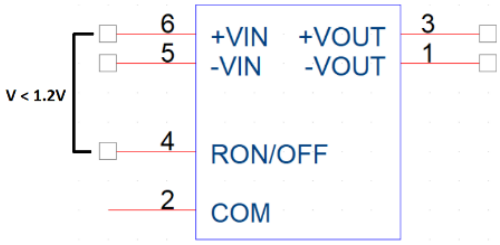


Fig. 4.6. Convertidor encendido (Método 2)

Este inconveniente supone que el puerto de control (R.C) que incorpora no puede controlarse a través del pin de una Raspberry Pi, ya que el voltaje que es capaz proporciona el pin es menor que el voltaje mínimo para controlar el encendido y apagado del convertidor. Esto supone un imprevisto en el diseño para esta parte del circuito.

La solución que se propone para solventar este problema es utilizar el primer método. Para ello, se decide añadir un transistor entre los puertos, $-V_{in}$ conectado al emisor y R.C donde se conecta el colector, que haga la función de interruptor. Este transistor trabaja en la zona de saturación, es decir, conduce o no corriente entre su de colector a emisor dependiendo si hay suficiente corriente aplicada en su base por el pin de la Raspberry Pi. Para regular la corriente que atraviesa la base se conecta una resistencia entre esta y el pin de la Pi.

Para el cálculo del valor de la resistencia se debe tener en cuenta la zona en la que trabaja el transistor deber ser en la zona de saturación para que este se comporte como un interruptor.

$$V_{pin} = 3.3V \quad (4.4)$$

$$Zona\ de\ saturación: Ic > \beta I_b \quad (4.5)$$

$$I_{BSATmax} = \frac{I_c}{10} = \frac{10mA}{10} = 1mA \;\; I_{BSATmin} = \frac{I_c}{50} = \frac{10mA}{50} = 200\mu A \quad (4.6)$$

$$R_{max} = \frac{V_{pin}}{I_{Bmin}} = 16.5k\Omega \;\; R_{min} = \frac{V_{pin}}{I_{Bmax}} = 3.3k\Omega \quad (4.7)$$

Entre estos valores de resistencia comerciales existentes, se decide escoger un valor de 10kΩ. Por lo tanto, la corriente de base resultante es:

$$I_{BSAT} = \frac{3.3V}{10\text{k}\Omega} = 330\mu A \qquad (4.8)$$

La corriente base se encuentra el intervalo para la saturación del transistor.

En caso de que el pin proporcione 3.3V, es decir, que esté configurado a nivel alto, el transistor conduce entre colector y emisor. Esto provoca un cortocircuito entre los puertos $-V_{in}$ y R.C, que como se ha mencionado anteriormente, produce el apagado del convertidor. Por el contrario, si se aplican 0V el transistor no conduce entre colector y emisor ($I_c = 0$), lo que provoca que el convertidor se encienda.

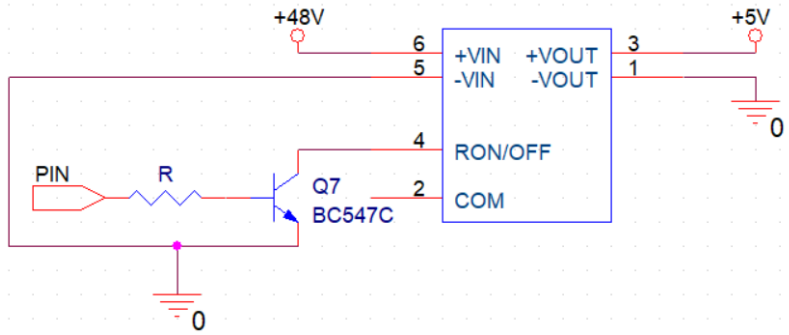


Fig. 4.7. Convertidor controlado mediante un transistor

Para estas especificaciones es suficiente con un transistor básico que soporte corrientes pequeñas de colector a emisor. Por lo tanto, se escoge un modelo común de transistor, un BC547B.

El control del apagado y encendido de uno de los convertidores, no nos permite controlar de forma individual cada dispositivo conectado. La funcionalidad que nos proporciona es la del control de la alimentación en todos los dispositivos conectados. Por otra parte, el segundo convertidor nos permite apagar la Raspberry Pi y el resto del circuito.

Los pines GPIO elegidos para el control de los convertidores positivos son:

- Convertidor para alimentar los dispositivos: GPIO18 (BCM 24).
- Convertidor para alimentar la Raspberry Pi y al circuito: GPIO15 (BCM 22).

El esquema de la conversión de tensión se muestra en la figura.

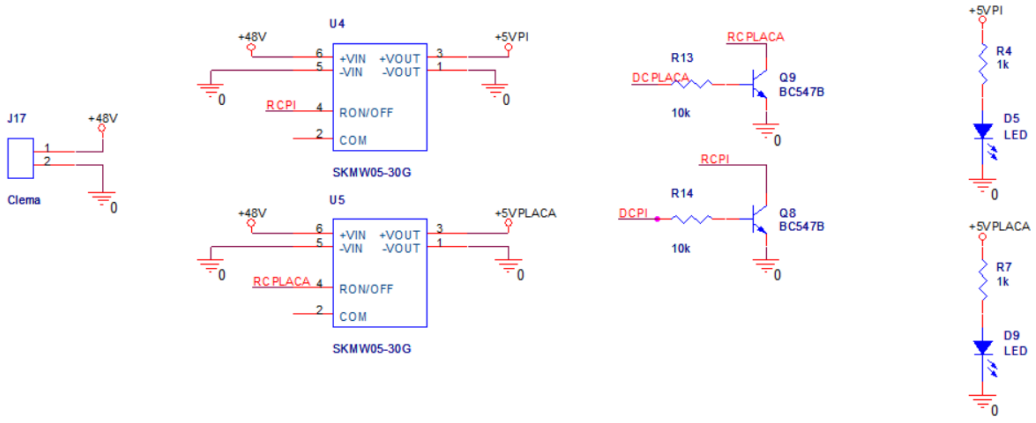


Fig. 4.8. Bloque de alimentación y conversión

Lista de componentes seleccionados:

- 2 resistencias de 10kΩ.
- 2 resistencias de 1kΩ.
- 2 leds: APHHS1005LQBC.
- 2 convertidores: SKMW30G-05.
- 1 clema de dos puertos: WAGO 235.
- 2 transistores: BC547C.

4.2.2.2 Bloque de control del dispositivo

Otro requisito que se define es controlar el encendido y apagado de los dispositivos de manera individual. Para llevarlo a cabo, existen una gran variedad de componentes en el mercado que permiten interrumpir la alimentación en un circuito eléctrico, los más comunes son el interruptor o el pulsador. Debido a que se manejan valores de corriente altos se escogen los relés, a pesar de que tengan la desventaja de tener una velocidad de conmutación más lenta y otras desventajas que se mencionaran más tarde. Por otra parte, al igual que ocurre con el transistor, permite el control remoto de un dispositivo con pequeñas tensiones de control.

El control como interruptor se produce a través de un pin de la Raspberry Pi, que proporciona una tensión de 3.3V provocando que el relé se desactive, y una tensión de 0V que, por el contrario, excita el relé. En función de la implementación que se desee, el relé permite la corriente el paso de C(Común) a NA (normalmente abierto), o de C a NC (normalmente cerrado). Con el objetivo de utilizar la alimentación proveniente del convertidor, el modelo de relé a escoger debe funcionar a una alimentación de 5V. El modelo escogido es un Finder 40.51 (consultar Anexo F) que es capaz de trabajar a la tensión requerida.

El problema del relé es que si se produce algún mal funcionamiento de un dispositivo conectado o un error de conexión puede dañar el resto del circuito incluida la Raspberry Pi, además ninguno de sus pines GPIO es capaz de proporcionar la corriente suficiente (máximo 3mA por pin GPIO) para hacer funcionar la bobina del relé . Por ello para proteger el sistema de control de cualquier ruido, sobrevoltaje o pico es necesario aislar el circuito del relé. Además, al propio relé debe ir conectado un diodo en antiparalelo de

manera que al interrumpir de manera brusca la corriente de la bobina que forma parte del relé, permita que esta se descargue de forma controlada evitando grandes picos de tensión. El modelo de diodo escogido para evitar estos problemas es el 1N4148, este diodo cumple con las especificaciones de protección. El esquema del relé se muestra en la siguiente figura.

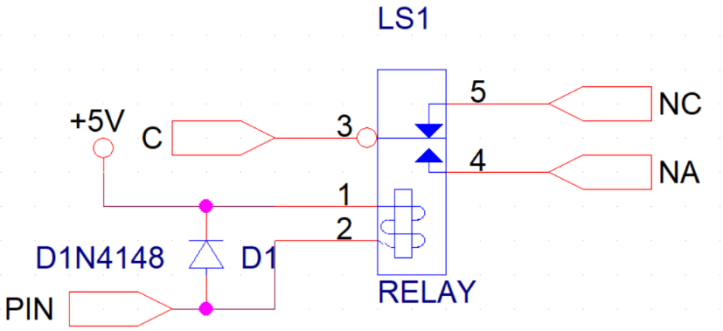


Fig. 4.9. Relé con un diodo de protección

Para aislar el circuito se decide utilizar un optoacoplador que es un componente formado por un diodo led y un fototransistor, lo que resulta que ambos elementos no estén conectados físicamente consiguiendo un aislamiento óptico.

Para hacer funcionar el optoacoplador es necesario que el modelo elegido funcione con una tensión de 5V que proporciona el convertidor. El modelo escogido en este caso es el TPC817C (Consultar Anexo C). Este modelo de optoacoplador es capaz de soportar una alimentación de 5V y tiene un alto valor de aislamiento óptico.

El diodo led del optoacoplador se conecta con el ánodo a 5V y con el cátodo a un pin GPIO de la Raspberry Pi. Este último, al igual que sucedía con el relé, controla el paso o no de corriente a la entrada del optoacoplador.

A continuación, se necesita conectar una resistencia a la entrada con el valor para que regular la corriente y evitar que se dañe el optoacoplador y la Raspberry Pi.

En esta situación el relé no funciona debido a que la salida del optoacoplador no es capaz de proporcionar la suficiente corriente para excitar la bobina de este que según sus hojas de características es de 90mA.

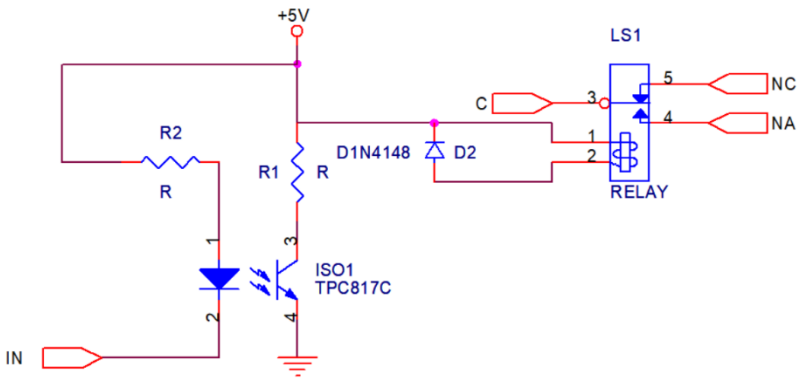


Fig. 4.10. Relé y optoacoplador

En hilo con lo anterior, para conocer qué conector de alimentación está en funcionamiento en el circuito, se necesita añadir un led a la entrada del optoacoplador. El led se comporta de tal manera que se enciende cuando el pin de la Rasberry Pi no conduce corriente y se apaga en caso contrario.

Junto con el led se ha de conectar una resistencia para regular la corriente a la recomendada por el fabricante. Para elegir el valor de la resistencia debemos tener en cuenta las siguientes características:

- I_F: $c$orriente de trabajo del led que posee el optoacoplador a su entrada.
- V_F: caída de voltaje que causa el led en el circuito.
- $V_{LEDseñ}$: caída de voltaje que causa el led de señalización en el circuito.
- $I_{LEDseñMAX}$: $c$orriente máxima de trabajo del led de señalización.

La corriente máxima que soporta el led de señalización es $I_{LEDseñMAX} = 30mA$. Siguiendo la regla anterior, la corriente del led del optoacoplador que se establece es una décima parte del máximo soportado por el led, así que se escoge el valor de 2.5mA.

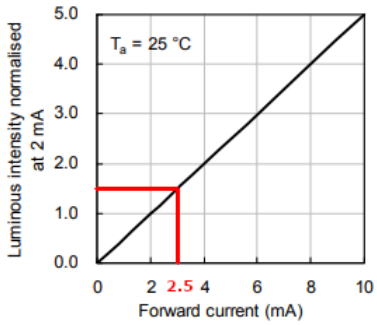


Fig. 4.11. Luminosidad vs corriente del led.

De acuerdo con la *Fig.4.11*, para una corriente de 2.5mA la caída de voltaje que produce el led de señalización en el circuito es de 2.6V.

El cálculo a realizar es el siguiente:

$$R_1 = \frac{5\,V - V_{LEDseñ} - V_{Fopt}}{I_F} = \frac{5\,V - 2.7 - 1.2}{2.5\,mA} = 440\Omega \cong 470\Omega \qquad (4.9)$$

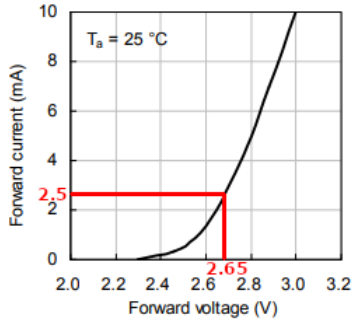


Fig. 4.12. Corriente del led vs Voltaje del led (Optoacoplador) (2)

Para una corriente de 2.5mA el CRT (coeficiente entre la eliminación del led y la corriente generada por el fototransistor) es del 100%. Por lo tanto, la corriente que se genera en el colector del fototransistor es:

$$I_c = I_F \; de \; 100\% = 2.5 \; mA \qquad (4.10)$$

Como se menciona anteriormente, la corriente que se genera en el fototransistor del optoacoplador es insuficiente como para hacer funcionar el relé, por lo que es necesario añadir un transistor en el circuito para solucionar este problema. Este transistor se conecta desde su base al emisor del fototransistor.

El colector del transistor se conecta directamente en serie a la bobina del relé. La corriente mínima necesaria para que funcione el relé a 5V según la hoja de especificaciones, es de 90mA. Por lo tanto, es un requisito indispensable que el modelo de transistor que se seleccione deba soportar al menos una corriente de colector máxima de un 10% más respecto a la corriente mínima del relé como margen de seguridad.

$$I_{cmax} = I_{cmin} + 10\% \; de \; I_{cmin} = 90 \; mA + 9 \; mA = 99 \; mA \qquad (4.11)$$

El modelo de transistor que se ha escogido para cumplir esta función es el mismo del apartado anterior, el BC547B, que tal y como se menciona en el Anexo D tiene la capacidad de soportar una corriente máxima de 100mA. El objetivo de escoger el mismo modelo que el utilizado en control de los convertidores, se traduce en un ahorro de costes en la selección de componentes.

El transistor debe funcionar en zona de saturación, debido a que en esta zona el transistor actúa como un conmutador (interruptor) permitiendo que la corriente en el colector sea igual a la corriente necesaria para que el relé funcione.

Por otro lado, cuando el diodo del optoacoplador se ilumina excita el fototransistor generando una corriente de 2.5 mA, provocando que su punto de trabajo se encuentre en la zona de saturación.

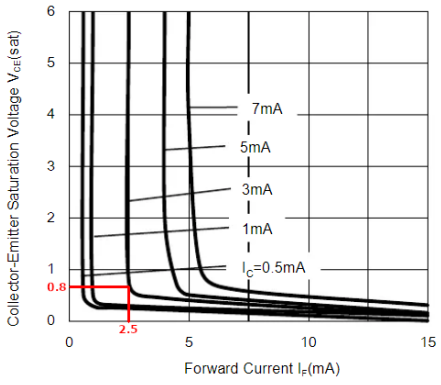


Fig. 4.13. Voltaje colector-emisor de saturación del fototransistor vs Corriente del diodo LED (optoacoplador)

A su vez, la corriente que se genera en el emisor del fototransistor atraviesa la base del transistor. Como en el caso del fototransistor, se requiere que el punto de trabajo se

encuentre en su zona de saturación, para que funcione como un interruptor. Con esto se consigue que la corriente que circula por el colector sea la adecuada para el funcionamiento del relé.

Como se ha mencionado anteriormente, según la hoja de características para que el transistor trabaje en zona de saturación la corriente de base deber ser entre 10 veces y 50 veces menor que la corriente que circule por el colector ($I_b = \frac{I_c}{10} \mid I_b = \frac{I_c}{50}$).

Para configurar el punto del trabajo del transistor en la zona de saturación, es necesario añadir una resistencia en la base del transistor (emisor del fototransistor), para poder regular la corriente al valor que se ha establecido según los cálculos.

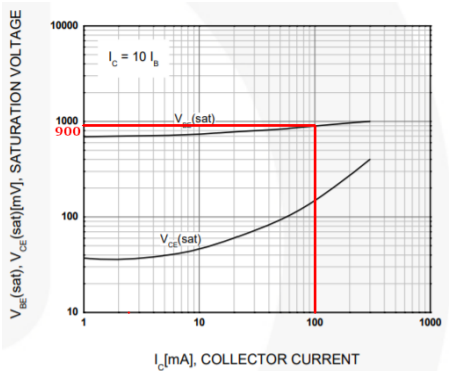


Fig. 4.14. Voltaje de saturación vs Corriente de colector

Partiendo de que el colector del fototransistor está conectado a una tensión de 5V (proveniente del convertidor), el voltaje de entrada en la base del transistor es:

$$V_{BEsat} = 0.9V \sim I_c = 100\ mA \qquad (4.12)$$

$$V_{CEsat} = 0.8V \sim I_F = 2.5\ mA \sim I_b = 2.5\ mA \qquad (4.13)$$

$$5\ V - V_{CEsatOPT} - V_{BEsatTRT1} = 5\ V - 0.8\ V - 0.9\ V = 3.3\ V \qquad (4.14)$$

El cálculo del valor de la resistencia queda de la siguiente manera:

$$I_b > I_{bSAT} \qquad (4.15)$$

$$\mathrm{I_{bSATmax}} = \frac{\mathrm{I_c}}{10} = \frac{100\ \mathrm{mA}}{10} = 10\ \mathrm{mA} \quad \mathrm{I_{bSATmin}} = \frac{\mathrm{I_c}}{50} = \frac{100\ \mathrm{mA}}{50} = 2\ \mathrm{mA} \qquad (4.16)$$

$$R_{min} = \frac{V_{in} - V_{CEsat}}{I_B} = \frac{3.3\ V}{5\ mA} = 660\ \Omega \quad R_{max} = \frac{V_{in} - V_{CEsat}}{I_B} = \frac{3.3\ V}{2\ mA} = 1.65\ k\Omega \qquad (4.17)$$

Entre estos valores de resistencia, el valor comercial escogido en este caso es 1 kΩ. La corriente base que circula entonces es:

$$I_b = \frac{3.3\ V}{1\ k\Omega} = 3.3\ mA \qquad (4.18)$$

Este valor de corriente es lo suficientemente alto como para que el transistor opere en la zona de saturación.

Por último, para conseguir un aislamiento completo entré el relé y la Raspberry Pi. Se añade un conector con dos terminales al circuito para separar la tensión que alimenta la parte del optoacoplador y el pin de la Raspberry Pi, con la parte del relé. Por lo tanto, se pueden conectar dos fuentes de tensión independientes para alimentar cada una de las partes. Como consecuencia, aumenta la seguridad frente a los problemas que puede provocar el relé en un circuito, como el ruido que genera la bobina. Sin embargo, a lo largo de este proyecto no se utilizará el aislamiento completo.

En este caso ambos terminales se conectan mediante un jumper con el objetivo de que ambas partes se alimenten de la misma fuente de tensión, la que proporciona el convertidor que también alimenta a la Raspberry.

El esquema final del bloque de control se muestra en la siguiente figura.

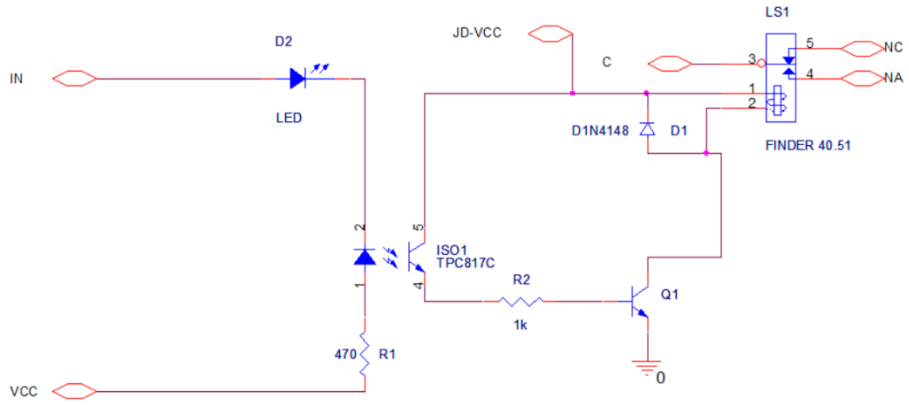


Fig. 4.15. Bloque de control

Debido a que el circuito debe soportar un máximo de cuatro dispositivos conectados, el bloque formado por el relé y el optoacoplador se implementa cuatro veces. A cada bloque optoacoplador-relé va conectado un pin GPIO de la Raspberry Pi, tal y como se ha explicado anteriormente.

Los pines GPIO elegidos para el control de los relés, es decir, para el control de los dispositivos son: GPIO 13 (BCM 27), GPIO 11 (BCM 17), GPIO 16 (BCM 23) y GPIO 31 (BCM 6).

El esquema del circuito implementado con los cuatro bloques se muestra en la siguiente figura.

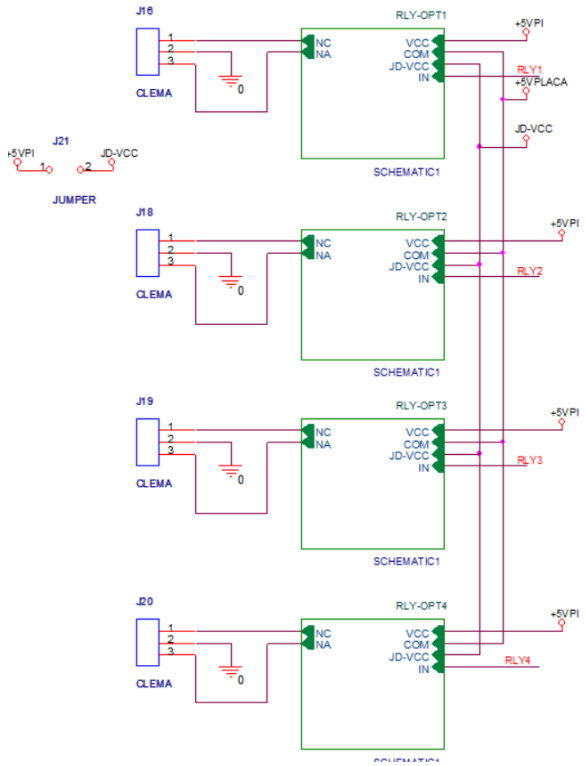


Fig. 4.16. Cuatro bloques de control conectados

Finalmente, los componentes escogidos para esta parte son:

- 4 relés: FINDER 40.51.
- 1 conector de dos pines.
- 4 transistores: BC547B.
- 4 leds: APHHS1005LQBC.
- 4 resistencias de 470 Ω.
- 4 resistencia de 1 kΩ.
- 4 diodos: 1N4148.

4.2.2.3 Bloque de conexión del dispositivo y la Raspberry Pi

La conexión de los dispositivos al circuito se hace mediante clemas, que es un tipo de conector que proporciona la flexibilidad y rapidez a la hora de conectar el dispositivo. De acuerdo con las especificaciones, se colocan un total de cuatro conectores con tres puertos cada uno.

Los dos puertos del extremo se conectan a cada una de las salidas de los relés, NA y NC, respectivamente, donde se conecta la tensión de 5V, mientras que el puerto del centro del conector se destina para la conexión de la masa del dispositivo. Se disponen las conexiones de esta manera para que la masa haga de aislador del ruido entre las dos salidas del relé. En la *Fig. 4.16* se muestra la disposición de las clemas.

Por otro lado, como se indican en los requisitos para conectar la Raspberry Pi a la placa se implementa un conector 2x20 pines. La conexión se lleva a cabo desde el ‘pinout’ de la Raspberry Pi a través de un cable de 40 pines.

En este conector no se mapean todos los pines, solo aquellos que se requieren como los pines de alimentación, los pines GPIO utilizados para el control y los pines reservados para los distintos protocolos de comunicación.

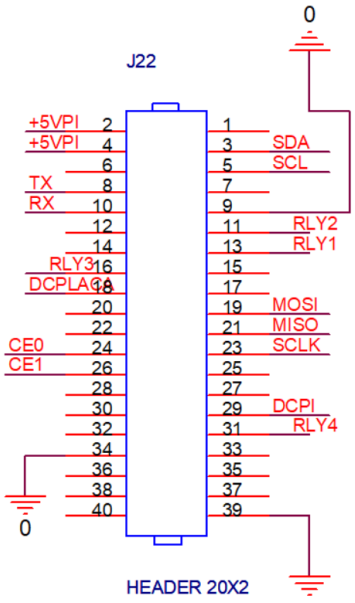


Fig. 4.17. Conector de 2x20 pines

Finalmente, los componentes escogidos para esta parte son:

- 4 clemas de tres puertos: WAGO 235.
- 1 conector de 2x20 pines.

4.2.2.4 Bloque comunicación con el dispositivo

Para el intercambio de datos con los dispositivos, se necesita implementar una serie de conectores físicos que permitan la conexión de las Zybo para la comunicación con los distintos protocolos especificados en los requisitos. Siguiendo el esquema de la siguiente figura.

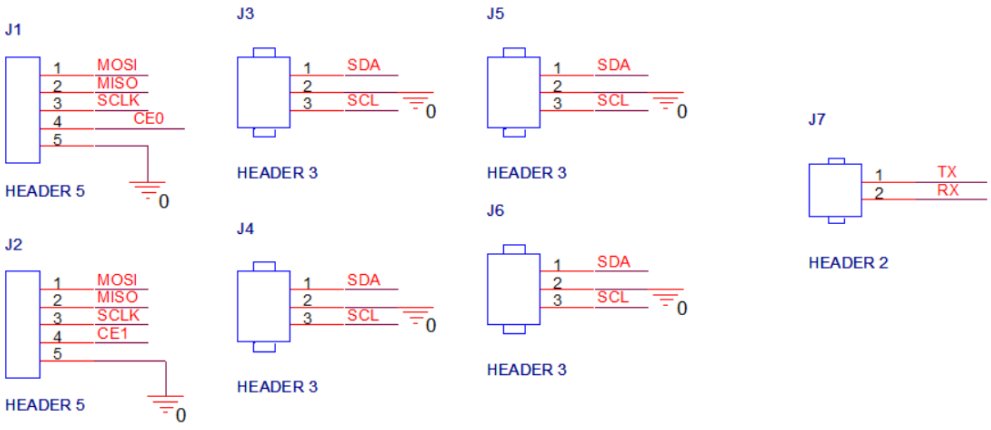


Fig. 4.18. Bloque de comunicación

Para comunicarse a través del protocolo serial USB-UART, la Raspberry Pi ya dispone de 4 puertos USB, uno por dispositivo por lo que no se implementa ningún conector en la placa del sistema de control.

En cambio, debido a que la Raspberry Pi implementa una interfaz UART que consta de dos pines GPIO, se añaden igualmente, un conector con dos terminales (J7). El objetivo de añadir este puerto serial adicional, no es únicamente el de dar soporte a otro dispositivo más sino también como puerto auxiliar, en caso de que alguno de los puertos USB tuvieran problemas.

En cuanto a la comunicación mediante el protocolo SPI, se disponen de dos conectores con cinco terminales (J1 y J2) el primer terminal es masa, el segundo es SSX, el tercero es CLK, el cuarto es MOSI y el quinto es MISO. La diferencia entre los dos conectores es que en un conector está conectada la línea SS0 para un dispositivo y en el otro el otro conector está conectada la línea SS1 para otro dispositivo diferente.

Finalmente, para la comunicación del protocolo I2C se implementan dos conectores de tres terminales (J3, J4, J5 y J6), el primero es SCL, el segundo es masa y el tercero es SDA.

Los detalles de cada protocolo se explican en apéndices posteriores.

Finalmente, los componentes escogidos para esta parte son:

- 2 conectores de cinco pines.
- 4 conectores de tres pines.
- 1 conector de dos pines.

4.2.2.5 Circuito completo

El esquema final del circuito de la placa del sistema de control se muestra en la siguiente figura.

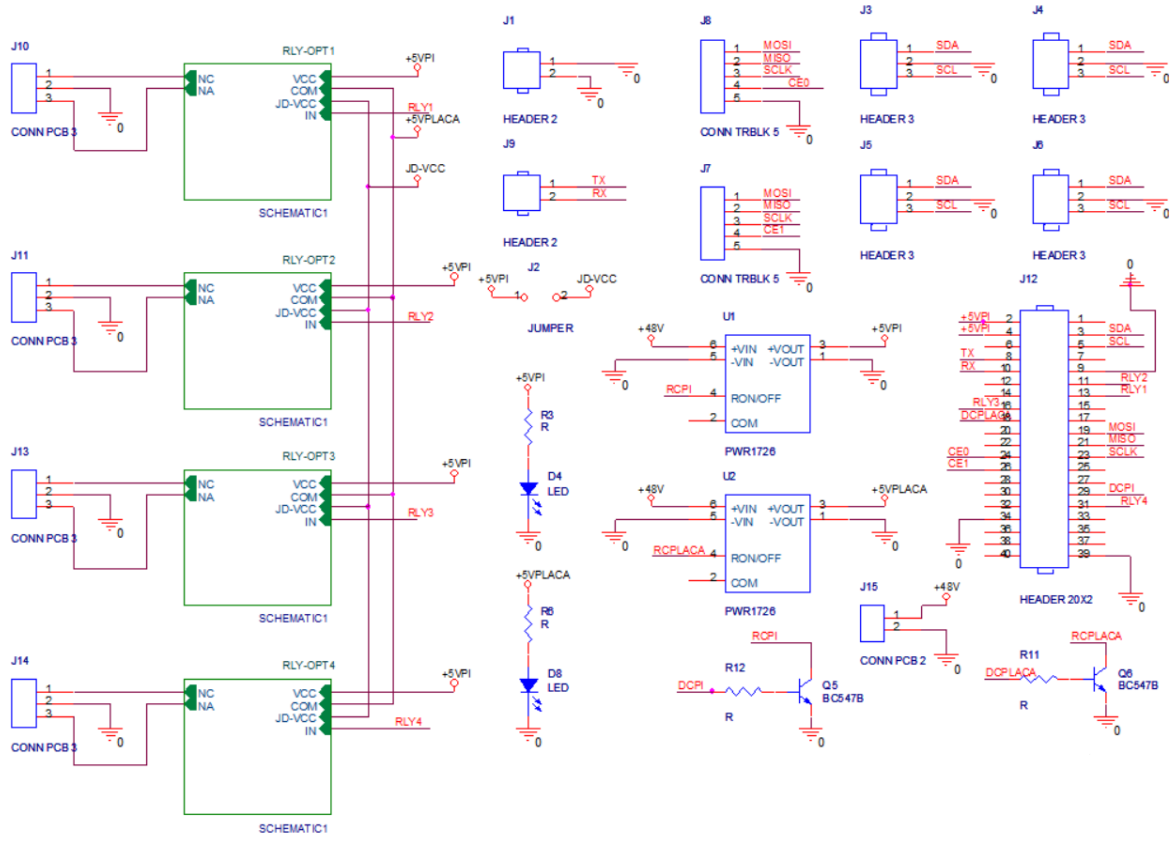


Fig. 4.19. Esquemático completo del circuito

4.3 Simulación en software especializado

Para tener una noción previa de cómo se comporta el circuito y de si está correctamente diseñado, se implementa los bloques más complejos en un simulador de circuitos.

El simulador de circuitos que se utiliza es Multisim (Anexo J), ya que tiene una forma más interactiva de mostrar el funcionamiento del circuito, a la hora de medir tensión y corriente en distintas partes del circuito.

El bloque se procede a simular es el de control, ya que es la parte de la placa de mayor complejidad de diseño y la que más cálculos aglutina.

La simulación consta de dos fases: la primera ejecuta las sentencias de funcionamiento al bloque individual de control, y la segunda fase consiste en simular cuatros bloques de control de manera simultánea.

4.3.1 Primera fase

En la primera fase se ha implementado el bloque de control de manera individual. La implementación se muestra en la figura.

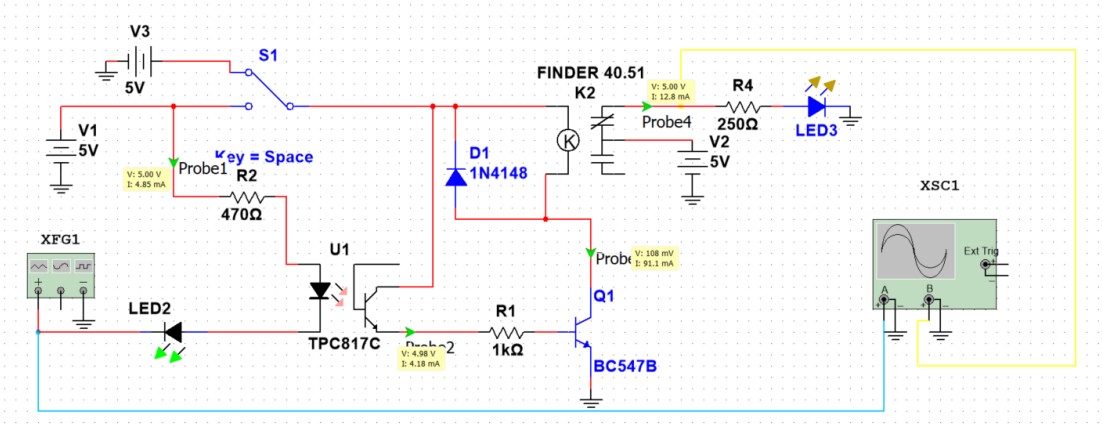


Fig. 4.20. Bloque de control implementado en Multisim

A la entrada del circuito se introducen los siguientes estímulos:

- Tres fuentes de tensión de 5V. La primera se conecta a la entrada del circuito para alimentar el optoacoplador y la segunda se conecta para alimentar el relé. En este sentido, ambas están controladas a través de un interruptor doble que simula la opción de aislamiento total que se ha implementado en el bloque de control. La tercera se utiliza para alimentar el led conectado a la salida del relé.
- Un generador de funciones que introduce una señal cuadrada de 0.825Vp y 0.825V de offset que imita el comportamiento de un pin de la Raspberry Pi a una frecuencia de 250mHz.

Por otro lado, al circuito se añade un led (con animación de iluminación) cuyo comportamiento indica si el relé conduce en NA o en NC de manera visual. En el mismo punto se añade un osciloscopio que compara la seña cuadrada generada en la entrada con la señal cuadrada que aparece en la salida.

Cabe destacar, que tanto el modelo de optoacoplador como el modelo de relé no es el seleccionado en el diseño, ya que no están incluidos en la base de datos de Multisim. A efectos prácticos algunos valores relacionados con el optoacoplador y el relé no pueden ser del todo fieles.

Para comprobar el funcionamiento del circuito se añaden una serie de nodos de prueba que miden la tensión y la corriente a lo largo del circuito. El objetivo de estos nodos de pruebas nos permite comparar los resultados de la simulación con los resultados teóricos.

El resultado de la simulación se muestra en las siguientes figuras donde se presenta una figura cuando el relé conduce por NC y cuando el relé conduce por NA.

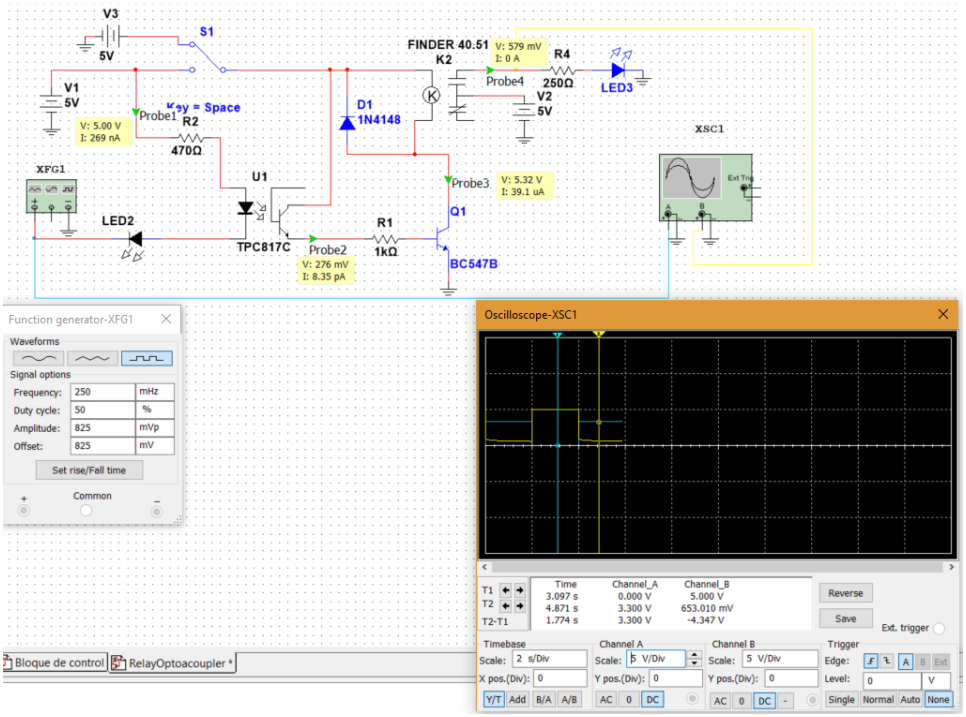


Fig. 4.21. Simulación del bloque de control (led apagado)

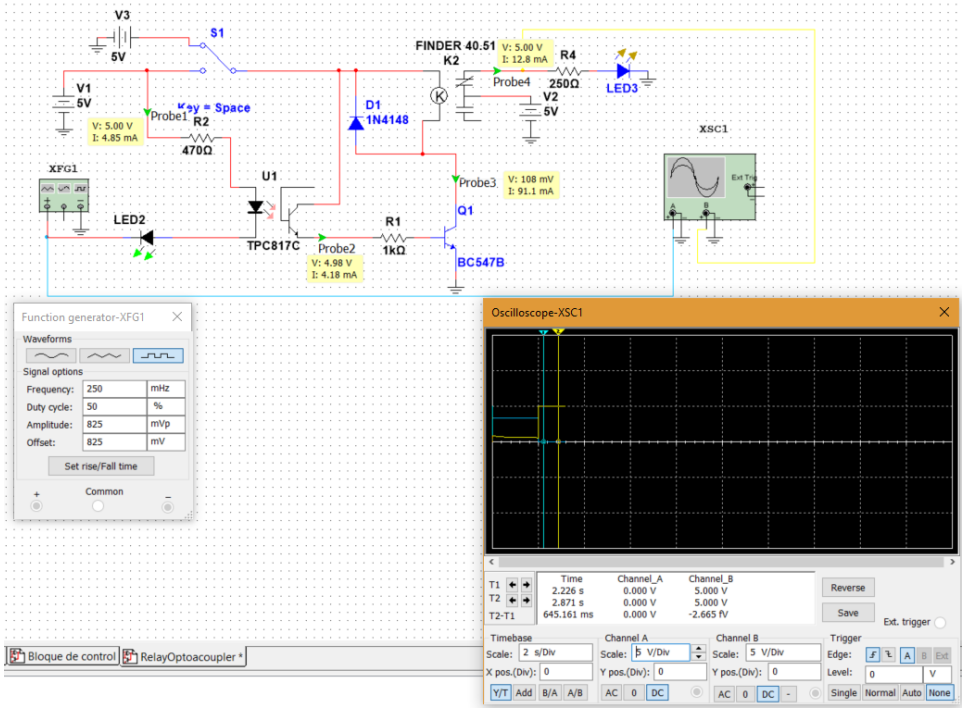


Fig. 4.22. Simulación del bloque de control (led encendido)

Se elabora una tabla para comparar los resultados teóricos y los obtenidos a través de los nodos de prueba en la simulación:

TABLA 4.1. COMPARACIÓN DE RESULTADOS (PRIMERA FASE)

	Teoría	Simulación

Prueba 1	5V~3.3mA	5V~4.5mA
Prueba 2	5V~3.3mA	5V~4.2mA
Prueba 3	5V~91mA	5V~91mA

Como se observa, lo resultados de la simulación virtual son muy parecidos a los que se han calculado en la parte teórica del proyecto.

Cuando la señal cuadrada toma su valor más bajo, se genera una corriente que atraviesa la entrada del optoacoplador suficiente como para iluminar el diodo led que excita al fototransistor. A su vez, la corriente generada por este último permite saturar el BC547B transistor para que la corriente que atraviesa el colector y emisor permita la activación del relé conduciendo la corriente por NA.

En conclusión, el circuito diseñado se comporta como esperamos.

4.3.2 Segunda fase

En la segunda fase se implementan los cuatro bloques de control como en el circuito diseñado y se conectan entre sí. La implementación se muestra en la siguiente figura.

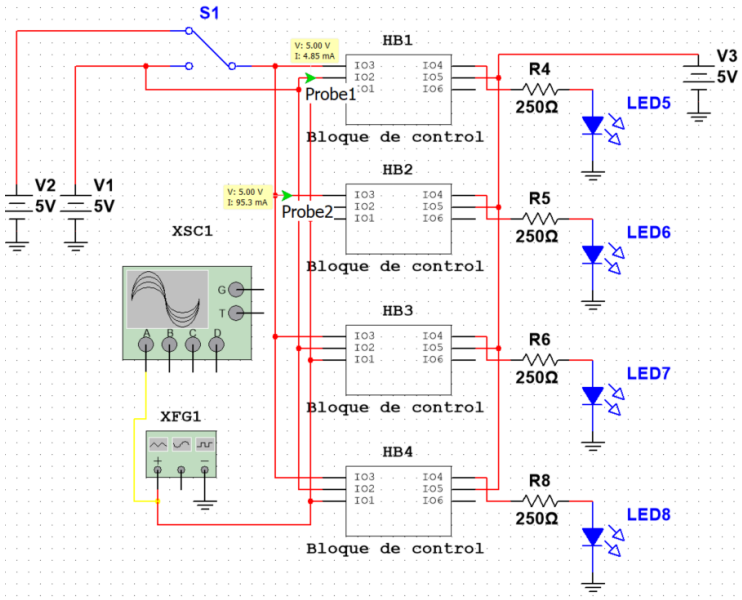


Fig. 4.23. Cuatro bloques de control conectados

A la entrada del circuito se introducen los siguientes estímulos:

- Tres fuentes de tensión de 5V. La primera se conecta a la entrada del circuito para alimentar el optoacoplador y la segunda se conecta para alimentar el relé. En este sentido, ambas están controladas a través de un interruptor doble que simula la opción de aislamiento total que se ha implementado en el bloque de control. La tercera se conecta para alimentar el led conectado a la salida del relé.

- Un generador de funciones que introduce una señal cuadradas de 0.825Vp y 0.825V de offset que imita el comportamiento de un pin de la Raspberry Pi a una frecuencia de 250mHz.

Por otro lado, se añade un led (con animación de iluminación) cuyo comportamiento indica si el relé conduce en NA o en NC de manera visual. En el mismo punto se añade un osciloscopio que visualizar la señal cuadrada generada en la entrada. Con esto y la visualización de los leds, se puede comprobar si el funcionamiento es el correcto.

Para comprobar el funcionamiento del circuito se añaden una serie de nodos de prueba que miden la tensión y la corriente a lo largo del circuito. El objetivo de estos nodos de pruebas es comparar los resultados simulados con los resultados teóricos.

El resultado de la simulación se muestra en las siguientes figuras donde se presenta en la *Fig. 4.24* cuando el relé conduce por NC y en la *Fig. 4.25* cuando el relé conduce por NA.

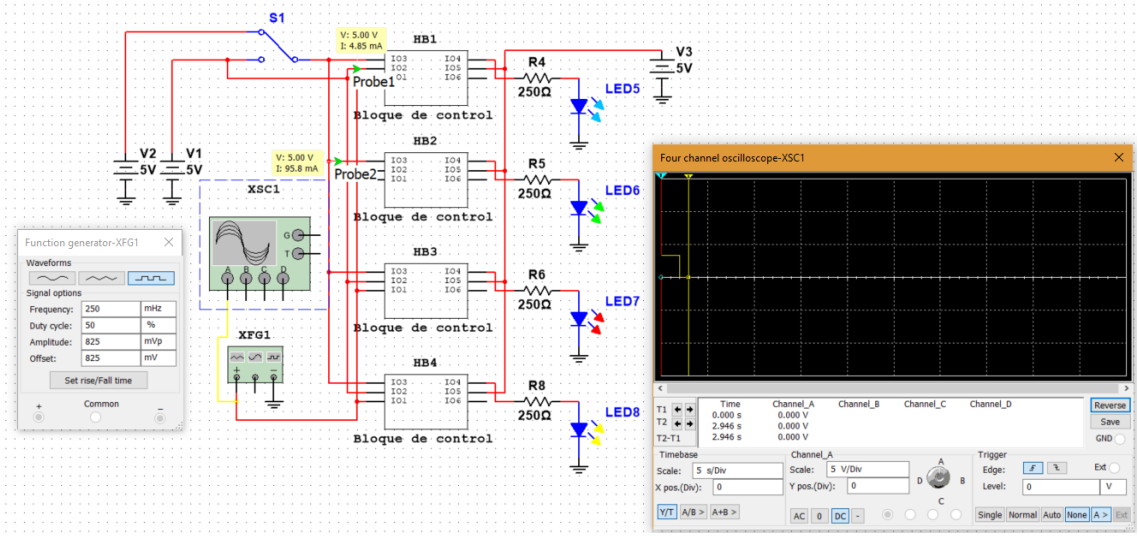


Fig. 4.24. Simulación del bloque de control (led encendido)

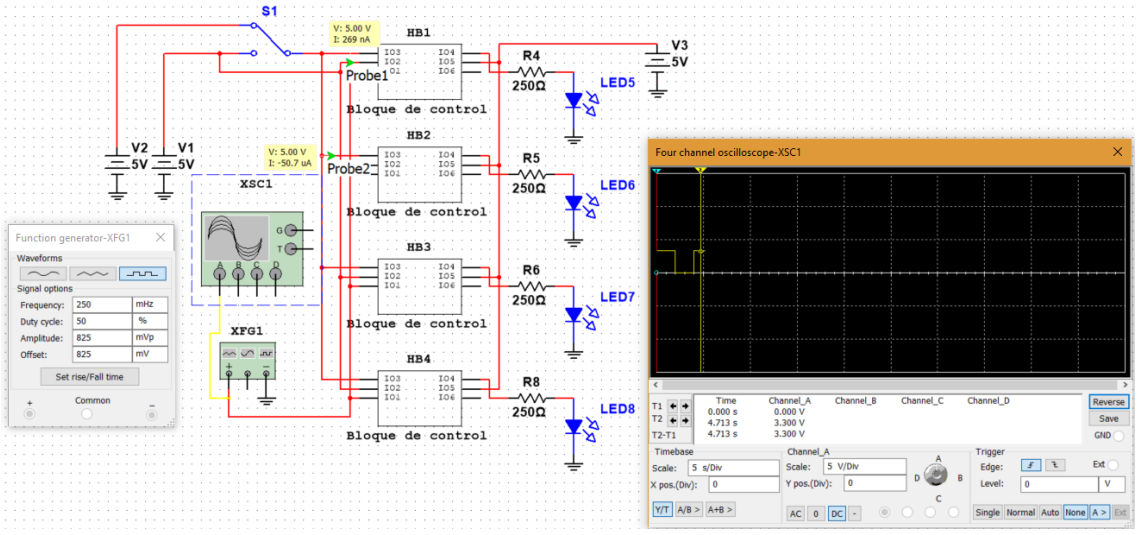


Fig. 4.25. Simulación del bloque de control (led apagado)

Se elabora una tabla para comparar los resultados teóricos y los obtenidos a través de los nodos de prueba en la simulación:

TABLA 4.2. COMPARACIÓN DE RESULTADOS (SEGUNDA FASE)

	Teoría	**Simulación**
Prueba 1	5V~3.3mA	5V~4.5mA
Prueba 2	5V~3.3mA	5V~4.2mA

Ante la vista de los resultados, cuando se conectan los cuatro bloques y funcionan de manera simultánea el comportamiento es el esperado. Tanto las tensiones como las corrientes que se observan en los nodos de se aproximan a los resultados teóricos y a los obtenido en la anterior simulación.

En conclusión, el diseño es adecuado para su implementación definitiva en el circuito que formará parte de la placa del sistema de control.

5 FABRICACIÓN DEL SISTEMA DE CONTROL

5.1 Introducción

Una vez elegidos los componentes a implementar y completado el diseño del esquemático del circuito de acuerdo con los requisitos establecidos, la siguiente fase es la fabricación física del circuito.

La fabricación consiste en implementar en un soporte físico, el circuito que hemos diseñado en la herramienta PSpice Capture CIS. La fase de fabricación consta de diferentes etapas: diseño y disposición de componentes en PCB, conexión entre componentes, perforación y soldadura.

5.2 Diseño y disposición de componentes

El soporte físico en el que se va a trasladar el esquema del circuito es una PCB.

Una PCB (Print Circuit Board) es una placa fabricada principalmente en cristales de vidrio donde se imprimen una serie caminos metalizados a modo de cables conocido como pistas, que interconectan los componentes y los sostiene mecánicamente. Para un mayor entendimiento en las explicaciones posteriores, conviene destacar que la unidad estándar que se utiliza como medida de longitud en una PCB son los ‘mils’ que en español significa milímetros de pulgada.

Para convertir el esquema del circuito en un esquema para PCB se va a utilizar la herramienta de conversación de esquemático a conversación de esquemático de circuito a esquemático en PCB que se encuentra en PSpice Capture CIS.

Una vez exportado, se importa a el programa OrCAD Layout Plus. Con esta herramienta, los componentes ya no son esquemáticos representativos que indican sus entradas y salidas, sino una forma denominada huella que representa el lugar donde se anclan los terminales del componente con sus medidas físicas reales. A partir de este momento para referirnos a medidas dentro del diseño de la placa utilizaremos la unidad ‘mils’ (milímetro de pulgada), excepto para las dimensiones absolutas de esta.

Cuando se importa el esquemático del circuito, se debe asociar a cada símbolo del componente una huella. En la siguiente tabla se muestra cada componente del circuito con su huella asociada:

TABLA 5.1. HUELLAS DE LA PLACA

Componente	Encapsulado	Huella
Relé	Agujero pasante	Anexo F
Optoacoplador	Agujero pasante	Anexo C
Resistencia	SMD	-
Transistor	Agujero pasante	Anexo D
LED	SMD	Anexo E

Clema de 3 puertos	Agujero pasante	Anexo H
Clema de 2 puertos	Agujero pasante	Anexo H
Tira de 3x1 pines	Agujero pasante	Anexo G
Tira de 5x1 pines	Agujero pasante	Anexo G
Tira de 2x1 pines	Agujero pasante	Anexo G
Tira de 2x20 pines	Agujero pasante	Anexo G

Para comenzar el diseño, se tiene que decidir la distribución de los componentes dentro de la placa. Una adecuada ubicación de los componentes se traduce en un circuito bien organizado que facilita el diseño de las pistas entre componentes.

En principio, para que el circuito sea lo más sencillo posible, se colocan los componentes agrupándolos en bloques de funcionamiento del mismo modo en el que se ha diseñado el circuito. Como consecuencia una ubicación eficiente de los componentes supone un ahorro de cantidad de cobre a la hora del trazado de las pistas.

En primer lugar, se define el tamaño de la base de fibra de vidrio. Se busca que el sistema de control en su conjunto (Raspberry Pi y PCB) sea manejable, y todo lo contenido posible.

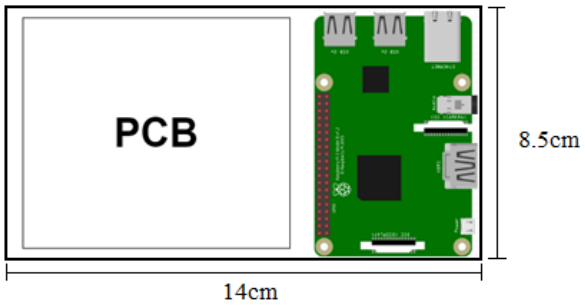


Fig. 5.1. Dimensiones del sistema de control

Con esto antecedentes, se establece que el alto de la PCB sea igual al alto de la Raspberry Pi que es de 8.5cm y el largo con un máximo de 14cm. Estas medidas se trasladan al programa de Layout Plus dibujando los límites de la placa.

Como hemos visto en el esquema de la *Fig. 5.1* la Rasberry Pi se decide colocar a la derecha del circuito. Esto supone que la tira de 2x20 (bloque de conexión dispositivos) pines donde se conectan los 40 pines de la Raspberry Pi se debe colocar en el lado derecho de la PCB, lo que nos facilita un mejor acceso para la conexión.

Siguiendo el sentido de derecha a izquierda de la *Fig. 5.1*, se ubica se ubica el bloque de alimentación y conversión del que forman parte los dos convertidores con su lógica de control y, en la parte de abajo, una clema de dos puertos a través de la cual se obtiene la tensión de 48V a convertir. Este bloque se ubica a una distancia preventiva de los

componentes más cercanos con el objetivo de evitar posibles averías en estos por un aumento excesivo de la temperatura de los convertidores.

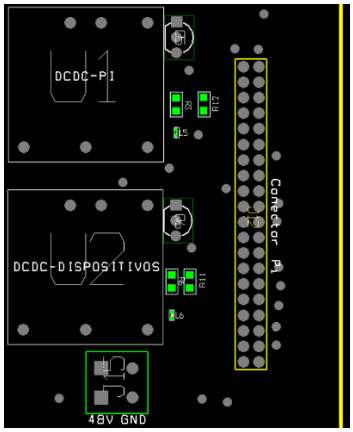


Fig. 5.2. Bloque de alimentación y conversión, y conector de 2x20 en la PCB

Por encima del bloque de alimentación y del conector de 2x20 pines, se sitúa el bloque de comunicación. En él se disponen de izquierda a derecha, cuatro conectores de 3 pines del protocolo I2C. A continuación, dos conectores de 5 pines del protocolo SPI y, por último, un conector de dos pines del protocolo UART.

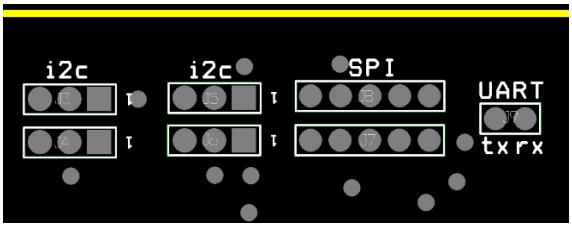


Fig. 5.3. Bloque de comunicación en la PCB

El propósito de esta ubicación facilita el acceso de la conexión y desconexión de la Zybo a los puertos físicos de los distintos protocolos.

A continuación del bloque de alimentación se coloca el bloque de control, formado por los cuatro conjuntos de optoacoplador-relé. La colocación de los componentes contenidos en este bloque es la misma para cada bloque de control.

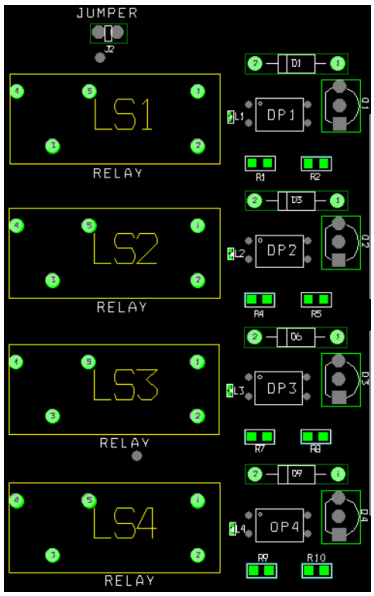


Fig. 5.4. Bloques de control en la PCB

Por último, se coloca la otra parte del bloque de conexión, las clemas de tres puertos. El objetivo de esta ubicación es debido a la cercanía que tienen con lo relés cuya salida es la alimentación de los dispositivos. Esto supone una mayor sencillez a la hora de diseñar las pistas, y un ahorro en la longitud de estas (ahorro de cobre).

Fig. 5.5. Clemas para la conexión de dispositivos en la PCB

La ubicación final de todos los componentes se muestra en la siguiente figura.

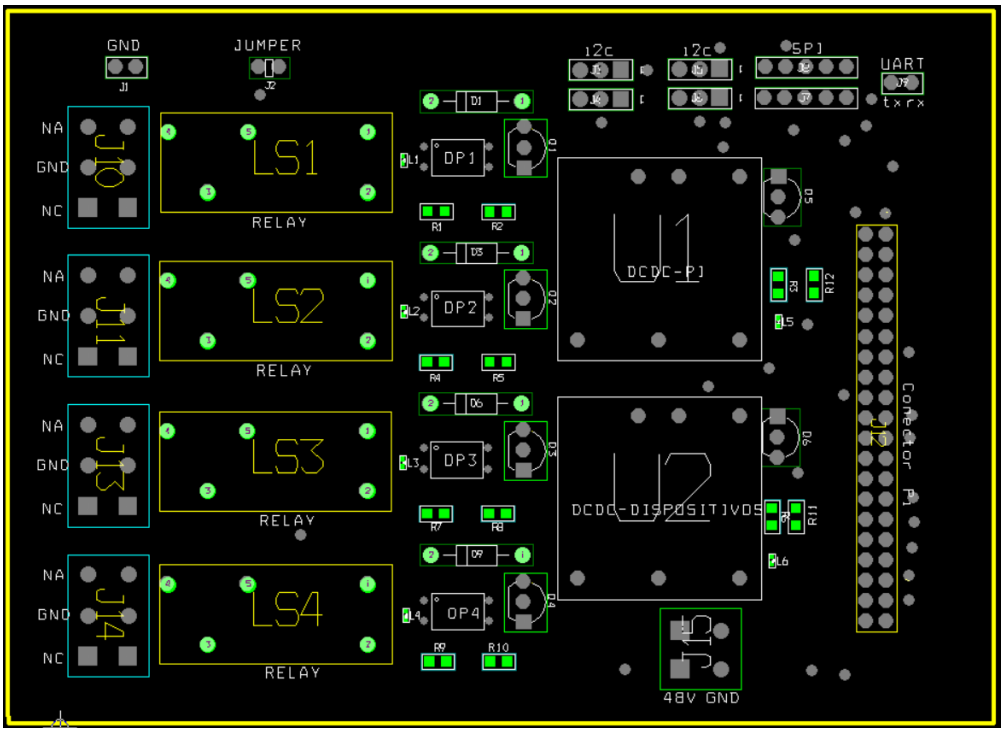


Fig. 5.6. Ubicación de los componentes en la PCB

Por otro lado, se necesita diseñar el tamaño de los pads que representan los terminales o pines de cada componente. El objetivo de esto es evitar dañar el pad una vez nos encontremos en el proceso de perforación, así como, poder disponer de la suficiente superficie de cobre para que en el proceso de soldadura el estaño se adhiera.

En concreto, la huella de cada componente ya implementa sus propios pads, pero para este caso no tienen el tamaño adecuado. La regla que se ha establecido para el tamaño es sea 2/1.5 veces el diámetro del agujero perforado en la fase de perforación. Este agujero se denomina ‘drill’ y a su vez, depende del tamaño del terminal del componente.

Se ha elaborado una tabla con los distintos componentes donde se detalla el tamaño del pad de acuerdo con el diámetro del agujero y el diámetro del terminal. La tabla es la siguiente.

TABLA 5.2. TAMAÑO DE LOS PADS POR COMPONENTE

Componentes	Tamaño del terminal (mils)	Diámetro del agujero (mils)	Tamaño del pad (mils)
Convertidor	49	80	160
Relé	15 x 40	80	160
Diodo	21	40	100
Optoacoplador	20 x 10	40	85
Resistencia	6 x 20	-	65 x 65

Transistor	22 x 20	40	80
LED	20 x 28	-	25 x 25
Clema	15 x 30	60	120
Tira de pines	25	50	85

Como se observa en la *Tabla 5.2*, con el led y la resistencia no se realizan agujeros porque son componentes SMD.

5.3 Diseño e impresión de pistas

Una vez distribuidos los componentes, se procede al diseño y trazado de las pistas que conectan los componentes de acuerdo con los cables que aparecen en el esquemático del circuito.

En un primer momento, con el objetivo de ahorrar costes, se intenta hacer un trazado por una sola cara. Sin embargo, se concluye que con el tamaño máximo de placa que se ha establecido en los requisitos, no es posible llevarlo a cabo.

Antes está problemática se plantean dos soluciones. La primera es hacer un diseño más largo de la PCB lo que supone mayor espacio entre componentes para el posterior trazado empeorando la manejabilidad del sistema y aumenta la complejidad de la distribución de las pistas y aumentan los costes

La segunda es enrutar en dos caras, la cara superior donde se posicionan los componentes y la inferior. Esto supone la ampliación del espacio de enrutado y disminuye la complejidad de la distribución de las pistas, aumenta la complejidad del soldado (necesidad de soldar por ambas caras) y aumentan los costes.

Se decide finalmente por utilizar un diseño a doble cara debido a la flexibilidad que nos proporciona imprimir las pistas por ambas caras, no afecta a la manejabilidad del sistema y debido a que los costes aumentando el tamaño de placa (para imprimir por una cara) son prácticamente iguales. Estos motivos se justifican en unos presupuestos elaborados que se muestran en la siguiente figura.

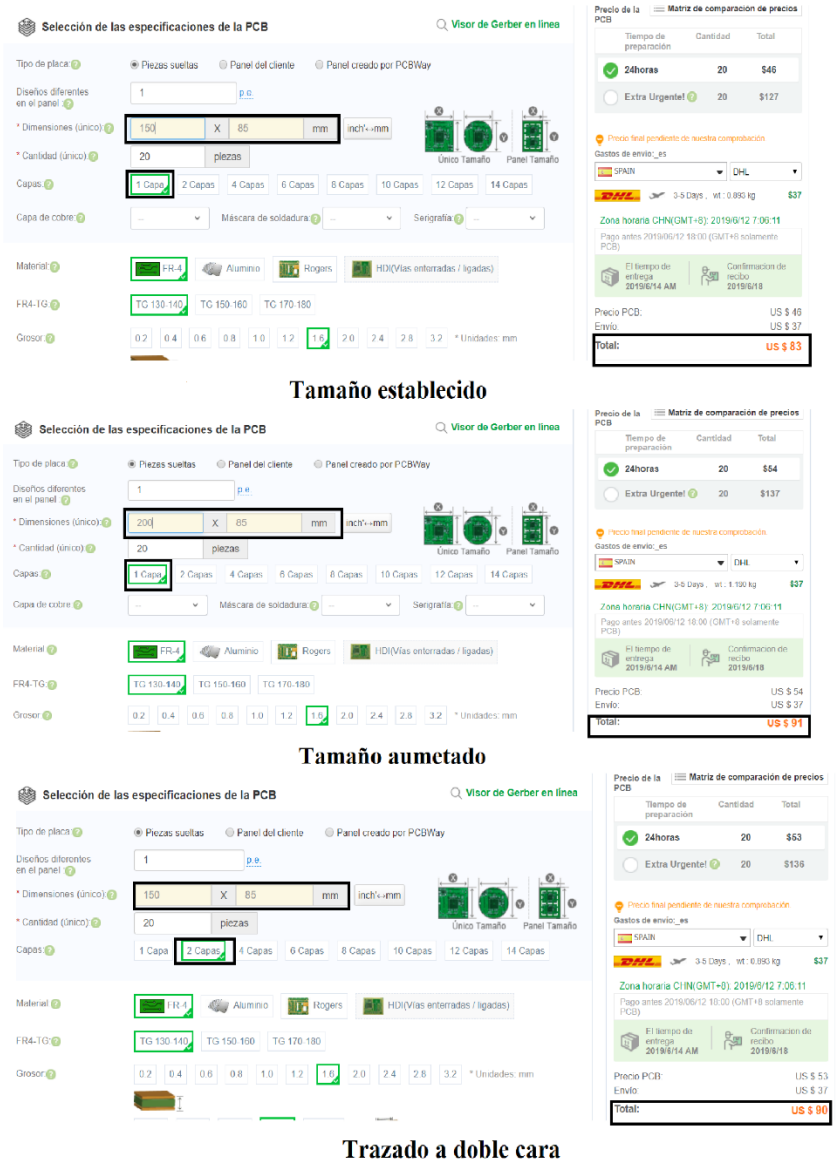


Fig. 5.7. Presupuesto ejemplo de fabricación de la placa

Cabe destacar que ante una solución de enrutado a doble capa y el uso de componentes con encapsulado DIP, una pista que se comienza trazando en la cara de arriba, no necesariamente debe terminarse en esta misma. Por implicaciones de diseño y espacio puede necesitarse cambiar de cara antes de conectar con otro pad. Como solución a esto se introduce el concepto de vía que son agujeros que permiten a una pista cambiar de una pista a otra.

Por otra parte, en el diseño de las pistas se deben configurar los siguientes parámetros ancho, grosor y longitud de la pista. Esto se debe a que estas características definen el comportamiento eléctrico de la pista. En este sentido, se debe tener en cuenta las limitaciones técnicas de la máquina que tiene la universidad para la fabricación de la PCB.

El ancho y el espesor afectan directamente a la intensidad máxima que es capaz de atravesar la pista. Mientras que la longitud es directamente proporcional a su resistencia, lo que provoca caídas de tensión.

Atendiendo a las limitaciones técnicas de la máquina, la pista debe tener como mínimo un ancho de 15-20mils ya que un valor menor no sería fabricable. Además, durante el proceso de impresión de la pista se suelen perder unos cuantos milímetros de pulgada.

Con ayuda de la herramienta online PCB Track Width Calculator (Anexo L) se calcula el ancho de la pista de acuerdo con la intensidad, el espesor y el aumento de la temperatura de la pista. Otro factor que se tiene en cuenta es si la pista es externa (al aire) o interna (por dentro de la placa). En este caso son todas externas.

Por lo tanto, se han establecido tres anchos diferentes de acuerdo con los siguientes parámetros comunes:

- Espesor: 2 $\frac{oz}{ft^2}$
- Aumento de temperatura: 10ºC

Se ha elaborado la siguiente tabla con los diferentes anchos de pista.

TABLA 5.3. ANCHO DE LAS PISTAS DE LA PLACA

Función	Corriente máxima (A)	Ancho de pista (mils)
Alimentación dispositivos externos	6	85
Alimentación Raspberry Pi y entrada de convertidores	2	55
Resto de conexiones	200m	30

Cabe destacar, que en los valores se ha dado cierto margen de seguridad por la reducción de varios milímetros de pulgada que se produce en el proceso de fabricación.

Por otro lado, es necesario establecer una serie de reglas que evitan, entre otras cosas, el solapamiento entre pistas, entre pista y pad, entre pads. Por lo tanto, se han definido las siguientes pautas:

- Pad a pad: 20mils.
- Pista a pista: 15mils.
- Pad a pista: 20mils.

Una vez establecido los requisitos para el proceso de enrutamiento, se procede a enrutar las pistas de acuerdo con el esquema del circuito de la figura.

El procedimiento de enrutado se plantea de manera que la pista sea lo más corta posible en cada conexión entre pads para reducir al máximo la caída de tensión.

El resultado final del enrutado se muestra en la figura.

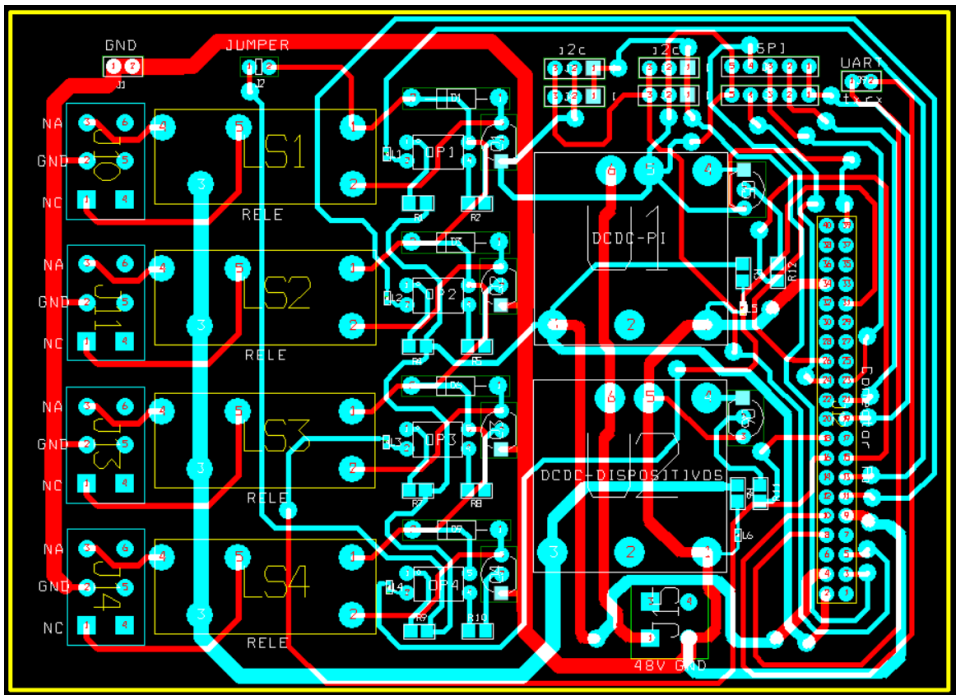


Fig. 5.8. Trazado final de la PCB

Con la herramienta Layout Plus se exporta los archivos con el diseño de la cara superior y la cara inferior para su fabricación.

En el proceso de fabricación el diseño de pads y pistas se imprimen en la máquina que se dispone en la universidad. Por lo tanto, resultante de la PCB es el que se muestra en la *Fig. 5.9.*

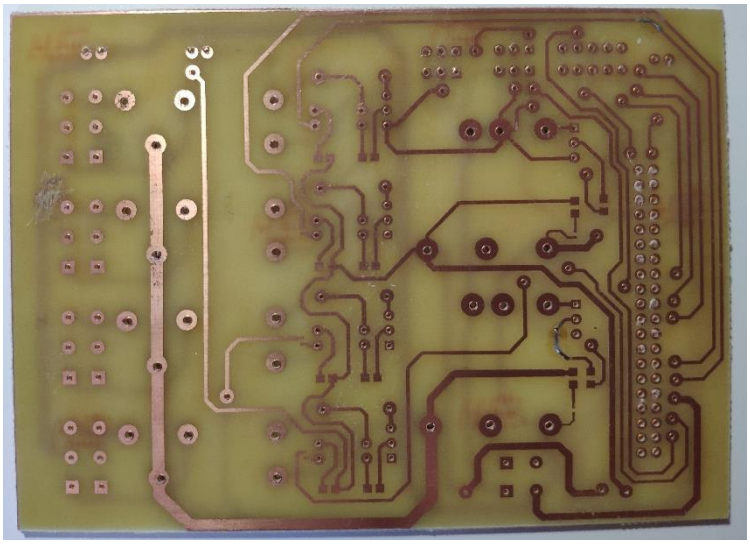


Fig. 5.9. Placa impresa (Capa superior)

5.4 Perforación

Una vez impresas las pistas y los pads en la placa para cada componente se inicia el proceso de perforación. Cómo se ha mencionado en los apartados anteriores para los

componentes con agujero pasante, se deben hacer agujeros para poder colocarlos en la placa.

El tamaño del agujero se realiza en función del grosor de las patas del componente. Por lo tanto, se establece que el agujero tenga un tamaño mínimo de 1.5 y 2 veces la medida de la parte más gruesa de la pata, ya que podemos encontrarnos con patas rectangulares, redondas o cuadradas. El tamaño del agujero utilizado para cada componente se muestra en la *Tabla 5.2* del apartado anterior.

Cabe señalar, que los componentes que poseen un encapsulado SMD no necesitan el perforado para su ubicación.

Finalmente, tras el perforado se procede a hacer una limpieza con alcohol isopropílico para eliminar las fibras de vidrio desprendidas de la placa durante este proceso.

5.5 Soldadura

Tras el proceso de perforación, el siguiente paso es la soldadura. Este consiste en ubicar los componentes en el sitio que les corresponde y aplicar estaño por medio de un soldador a los pads y patas de este para fijarlo a la placa y conectarlo a las pistas.

Para facilitar el proceso de soldado, se establece un orden al soldar. Primero se soldarán los componentes SMD y luego los componentes de agujero pasante. En este sentido, se lleva a cabo el proceso de dentro hacia fuera, es decir, ante mismo tipo de encapsulado se sueldan antes los que más cerca estén del centro de la placa.

El principal motivo de establecer este orden es que algunos componentes de agujero pasante son muy voluminosos y dificultan el acceso posterior a componentes más pequeños como los SMD.

Para finalizar, se aplica alcohol isopropílico para limpiar el flux en el circuito que forma parte del hilo de estaño al fundirse.

5.6 Pruebas

Una vez terminado la fabricación de la PCB se procede a realizar una serie de pruebas para garantizar que el funcionamiento de la placa es el esperado. Las pruebas dividen en pruebas de conexión física y pruebas de funcionamiento.

5.6.1 Pruebas de conexión física

Se llevan a cabo una serie de pruebas de conexión durante y después del proceso de fabricación de la PCB. Estas se dividen en dos fases: comprobación tras el impreso de las pistas y comprobación una vez soldados los componentes.

La primera fase consiste en la comprobación de las conexiones entre pads de la PCB. El objetivo es asegurarnos de que las conexiones impresas en la PCB coinciden con las implementadas en el diseño. En este sentido, con la ayuda de un multímetro en modo continuidad, se comprueban si existen pistas abiertas o cortocircuitos entre pistas y entre pista-pad.

La segunda fase de las pruebas se realiza una vez soldados todos los componentes. En esta, como en la prueba anterior, nos ayudamos de un multímetro en modo continuidad para comprobar desde cada terminal del componente la pista a la que debe estar soldada. El objetivo es asegurarnos de que todo está bien soldado y conectado.

Cabe destacar que las dos fases se aplican en ambas caras de la PCB.

5.6.2 Pruebas de funcionamiento

Las pruebas de funcionamiento que se llevan a cabo son las siguientes: funcionamiento de los convertidores, conexión y alimentación de la Raspberry Pi, conexión, alimentación y control de una Zybo, conexión, alimentación y control de cuatro Zybo.

Las pruebas que se llevan a cabo son incrementales, es decir, se van añadiendo dispositivos en cada prueba que finaliza de manera satisfactoria. Cabe añadir que, al realizar una nueva conexión de la Raspberry Pi o de algún dispositivo, se desconecta la fuente de tensión previamente por seguridad.

Por otro lado, se añade un jumper al conector que completa el aislamiento óptico entre relé y Raspberry Pi, para utilizar directamente la alimentación que proviene del convertidor, y no la de dos fuentes independientes.

Por último, al inicio del documento se indica que la alimentación de 48V proviene de switch PoE (power over Ethernet). Con el objetivo de hacer las pruebas con mayor comodidad durante el desarrollo de estas se sustituye el switch por un trasformador de 48V.

5.6.2.1 Prueba de funcionamiento de los convertidores

La prueba realizada para comprobar el funcionamiento de los convertidores tiene como objetivo como su propia designación indica revisar el funcionamiento de todas las entradas y salidas de tensión de la placa. Así como, la tensión que permite funcionar a relés y optoacopladores. Gracias al conector de dos pines conectado a masa del que se dispone en la placa, facilita las medidas de voltaje.

En lo que respecta al montaje de la prueba, únicamente se conecta la fuente de alimentación de 48V a la clema de alimentación de la placa a través de un conector jack. Este conector posee dos cables, el positivo debe conectarse en el puerto izquierdo de la clema y el negativo en el segundo, la masa.

Cuando se conecta la fuente de alimentación, si no existe ningún problema los leds asociados a los convertidores se encienden. Con el objetivo de asegurarse de que la tensión es de 5V utilizamos un voltímetro que comprueba el voltaje a la salida de los convertidores, a la entrada del bloque de control, en los pines del conector que proporcionan la alimentación a la Raspberry (asegurándonos de que sólo los pines de 5V tienen está tensión) y, por último, en las clemas donde se conectan los dispositivos. El resultado obtenido es en cada una de estas zonas del circuito es de 5V.

El montaje de la prueba se muestra en la siguiente figura.

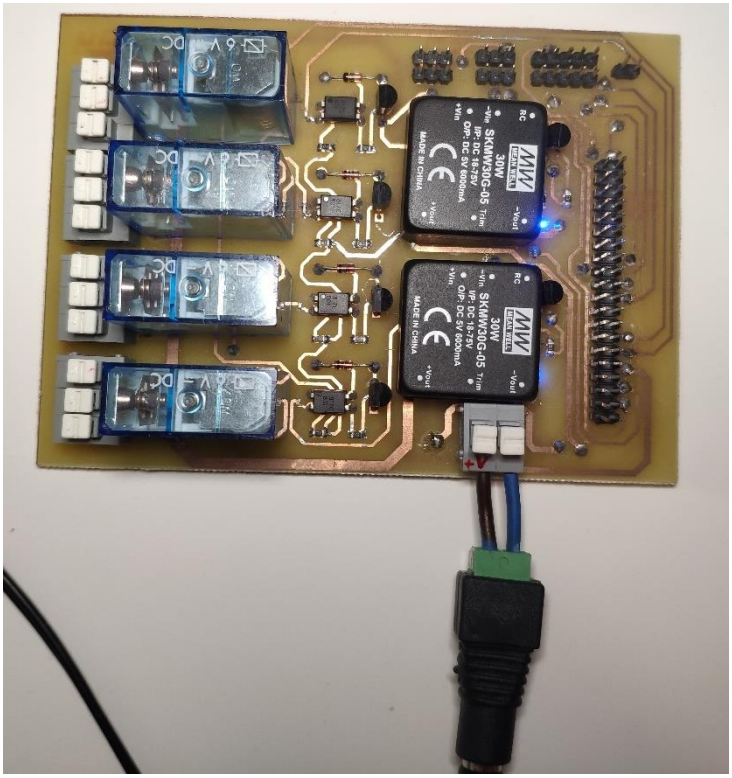


Fig. 5.10. Prueba del funcionamiento de los convertidores

5.6.2.2 Prueba de conexión y alimentación de la Raspberry Pi.

El objetivo de llevar a cabo esta prueba es el de detectar problemas de alimentación en la Raspberry Pi, y comprobar el funcionamiento del control a través del pin GPIO15 que se ha implementado sobre el convertidor del que obtiene su alimentación.

El montaje que se realiza consiste en la conexión de la Raspberry Pi a través de un cable de 40 pines desde su 'pinout' hasta el conector 2x20 pines implementado en la placa, y la conexión de un cable Ethernet para comunicarse con ella a través del ordenador.

Para comenzar con la prueba, se conecta la fuente a la placa a través de la clema de alimentación, y a continuación comprobamos si los leds de la Raspberry Pi están encendidos. Para garantizar que está funcionando correctamente, abrimos una conexión SSH con PuTTY y comprobamos que recibimos la petición de las credenciales de inicio de sesión de Raspbian.

En este caso se reciben estas credenciales, así que la potencia proporcionada es correcta. Sin embargo, en el caso contrario se tendría un problema de alimentación que conllevaría a investigar los factores que justificaran esta pérdida de potencia. Entre ellos se podría encontrar una mala soldadura en algún componente, un ancho de pista insuficiente o un déficit de potencia en el convertidor.

Por último, para comprobar el control remoto del convertidor que proporciona la alimentación a la Raspberry Pi. Con la ayuda de la librería ‘gpio’, a través de la sesión abierta en PuTTY se introduce los siguientes comandos:

gpio -g mode 22 out

gpio -g write 22 1

El primer comando configura el pin como salida y el segundo establece su valor a nivel alto. El comportamiento que se observa es el reinicio de la Raspberry Pi como consecuencia de la interrupción durante un instante de su alimentación. Es, por tanto, el comportamiento esperado.

El montaje de la prueba se muestra en la siguiente figura.

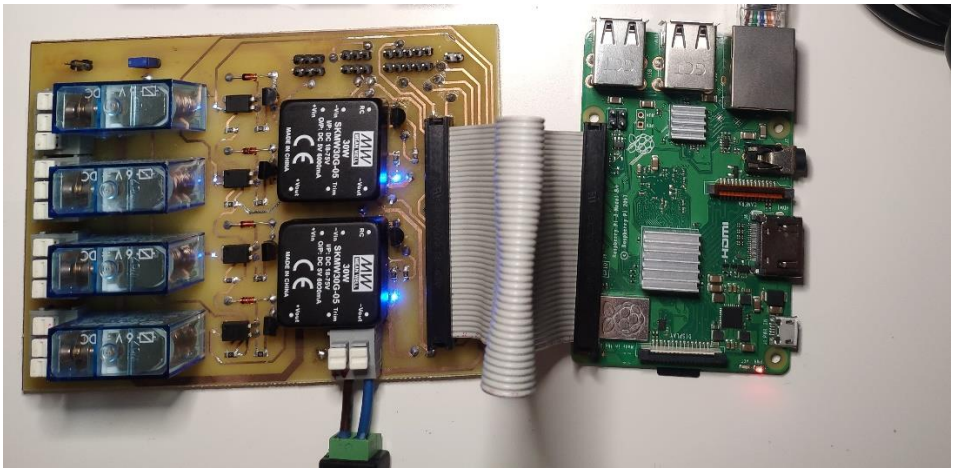


Fig. 5.11. Pruebas de alimentación de la Raspberry Pi.

5.6.2.3 Prueba de conexión, alimentación y control de una Zybo

El objetivo de esta prueba, como el caso anterior, es detectar problemas de alimentación en la Zybo, comprobar el control del convertidor asociado a la alimentación de las Zybo y comprobar el control de su alimentación de manera individual a través de los relés

En esta prueba del dispositivo, se conecta una Zybo a la primera clema (superior izquierda) conectada al relé gobernado por el pin GPIO13.

En la parte de la Zybo se configura el jumper para recibir la alimentación a través del puerto jack. Por otro lado, en la placa el cable positivo se conecta en el puerto superior de la clema y el cable de masa al siguiente.

Se conecta la fuente a la placa, se encienden los convertidores y la Raspberry Pi. En este momento la Zybo sigue apagada porque está conectada en el puerto correspondiente al terminal NA (normalmente abierto) del relé.

Como se ha procedido en la prueba previa, ejecutamos PuTTY para establecer una conexión SSH e introducir las credenciales de inicio de sesión. A través de PuTTY ejecutamos el siguiente comando:

gpio -g mode 27 out

gpio -g write 27 0

El primer comando configura el pin como salida y el segundo establece su valor a nivel bajo.

Como consecuencia, la corriente excita la bobina del relé provocando que el interruptor se cierre en el terminal NA permitiendo el paso de la corriente a través de él. En este caso, se observa que el led asociado al funcionamiento del relé y la Zybo se encienden.

Por el contrario, en el caso supuesto de que no se encendiera la Zybo las causas de este problema pueden ser, descartando problemas en la Zybo o en la Raspberry Pi, una soldadura deficiente en algún componente relacionado, un ancho de pista insuficiente o un déficit de potencia en el convertidor o un problema en el bloque de control.

Por último, se comprueba el control remoto del convertidor asociado a la alimentación de las Zybo. Se procede de la misma forma anterior, a través de la sesión abierta de PuTTY introducimos el siguiente comando:

gpio -g mode 24 out

gpio -g write 24 1

Como se ha mencionado anteriormente, el pin GPIO18 se configura como salida y se establece a nivel alto. Se observa que el led del convertidor se apagar indicando que no hay tensión a su salida. En consecuencia, la Zybo se apaga.

El montaje de la prueba se muestra en la siguiente figura.

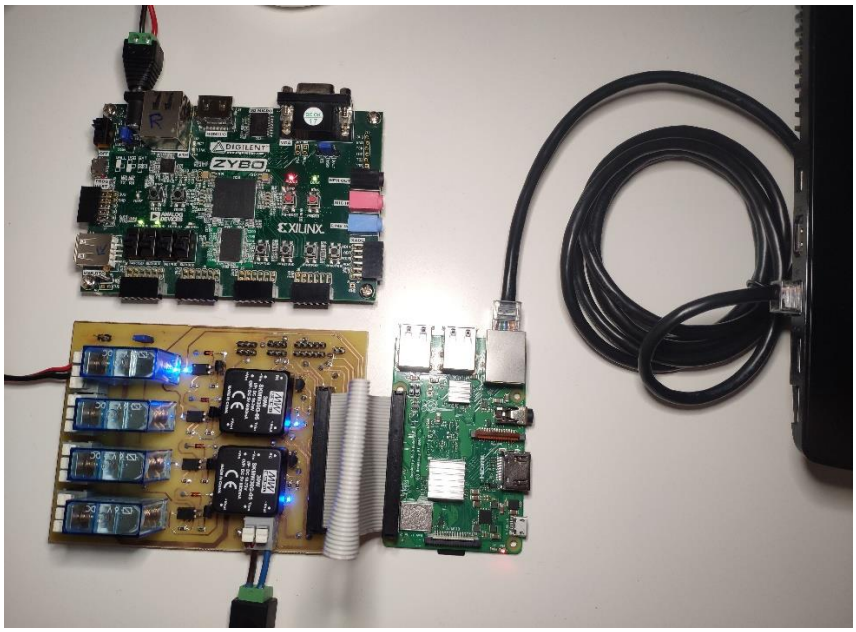


Fig. 5.12. Pruebas de conexión, alimentación y control de una Zybo

5.6.2.4 Prueba de conexión, alimentación y control de cuatro Zybo

Con esta prueba, el principal propósito que se persigue es detectar posibles fallas de potencia al tratar de alimentar las cuatro Zybo de manera simultánea. Además, se repiten las pruebas llevadas a cabo en el apartado anterior, tanto la comprobación del control individual de las Zybo por medio de los relés como el control de la alimentación que proporciona el convertidor asociado a ellas.

Los dispositivos utilizados es un conjunto de cuatro Zybo dispuestas una encima de otra que se conectan al sistema de control (donde ya está previamente conectada la Raspberry Pi).

En el lado del sistema de control, el montaje de esta prueba consiste en conectar cada Zybo a cada una de las clemas que se disponen en la placa a través de un par de cables de alimentación terminado en un conector jack macho. El cable positivo se conecta en el puerto superior de la clema y el cable de masa al siguiente. De este modo, quedan conectadas al puerto NA que impide el paso de corriente a través de ellas.

En el lado de la Zybo, cada una de ellas está configurada con el jumper para recibir la alimentación a través del puerto jack.

En primera instancia, el procedimiento de la prueba consiste en conectar la fuente a la placa, encendiéndose los convertidores y la Raspberry Pi. A continuación, ejecutamos PuTTY para establecer una conexión SSH e introducir las credenciales de inicio de sesión. En este momento la Zybo sigue apagada porque está conectada en el puerto NA.

Una vez preparado y alimentado el sistema de control se procede a probar la alimentación de las cuatro Zybo de manera simultánea y el control de cada una de ellas a través de los relés.

Por seguridad, se evita activar las Zybo de manera simultánea para no generar un pico de corriente que active la protección que contiene el propio convertidor. Por lo tanto, en esta prueba en vez activar los pines GPIO mediante línea de comandos, se desarrolla un pequeño script con tiempo de espera que configura los pines GPIO13, GPIO11 , GPIO16 y GPIO31 como salida, y cambia sus niveles lógicos tras un cierto tiempo de espera en un bucle infinito.

Como consecuencia, se observa que el led asociado al funcionamiento del relé y la Zybo se encienden indistintamente dependiendo del estado lógico de cada pin.

Por el contrario, en el caso supuesto de que no se encendieran las Zybo las causas de este problema podrían ser, descartando problemas en las Zybo o en la Raspberry Pi, una soldadura deficiente en algún componente relacionado, un ancho de pista insuficiente o un déficit de potencia en el convertidor o un problema en alguno de los bloques de control.

Por último, se comprueba el control remoto del convertidor asociado a la alimentación de las cuatro Zybo. A través de la sesión abierta de PuTTY anteriormente, introducimos el siguiente comando:

```
gpio -g mode 24 out

gpio -g write 24 1
```

Estos comandos configuran el pin GPIO18 como salida y se establece a nivel alto. Se observa que el led del convertidor se apagar indicando que no hay tensión a su salida. En consecuencia, la cuatro Zybo se apagan de manera simultánea.

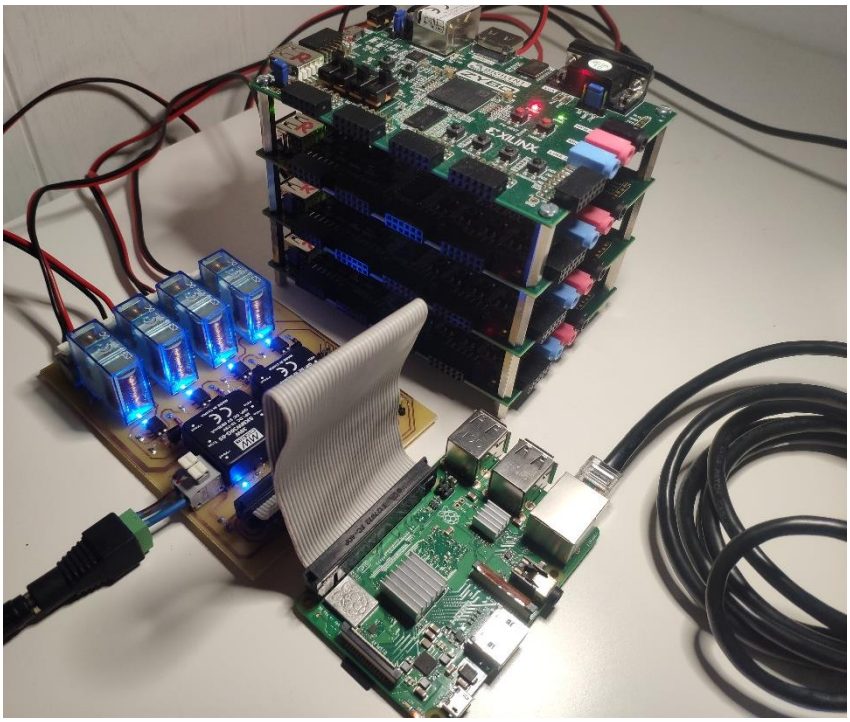


Fig. 5.13. Prueba de conexión, alimentación y control de cuatro Zybo

6 DISEÑO DEL SOFTWARE PARA EL SISTEMA DE CONTROL

6.1 Introducción

En este tema, se detallan los recursos y procedimientos necesarios para implementar la parte software del bloque de comunicación del proyecto. Como se ha mencionado al principio del documento, el objetivo de este bloque es intercambiar los datos recogidos por las Zybo y el sistema de control. Asimismo, dentro de este bloque se incluye la lógica del control de apagado y encendido automático de las mismas.

En primer lugar, se explican los diferentes protocolos que se van a implementar de acuerdo con los requisitos establecidos al inicio del documento. De este modo, se definen la comunicación serial, que protocolos se pueden implementar, como funcionan, y se señalan sus fortalezas y debilidades. Asimismo, se dan las explicaciones teóricas oportunas para justificar el uso de protocolos serial junto con cada uno de sus tipos.

En segundo lugar, a nivel de desarrollo software se describen los lenguajes de programación utilizados para la implementación de los distintos protocolos serial utilizados tanto a la hora de programar las FGPA como a la hora de programar la Raspberry Pi. Donde se detalla los requisitos, la estructura del desarrollo, librerías de relevancia utilizadas, el funcionamiento y el resultado de las pruebas del programa. En este sentido, cada protocolo se divide la parte desarrollada en las Zybo con la parte desarrollada en la Raspberry Pi.

6.2 Comunicación serial

La comunicación serie es procedimiento que se realiza para enviar datos de un sitio a otro mediante bits, utilizando como conexión física un bus o canal. Como su propio nombre indica los datos en esta comunicación se envían de manera secuencial, tal y como se muestra en la siguiente figura.

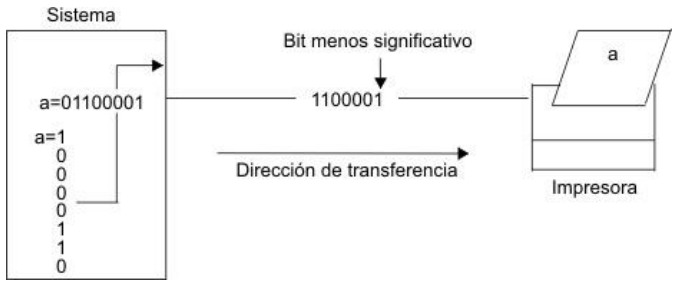


Fig. 6.1. Ejemplo de comunicación serial[11]

Por otro lado, estaría la comunicación paralela, cuya principal diferencia es que todos los bits de información se envían a la vez, es decir, debe existir un canal por cada bit transmitido.

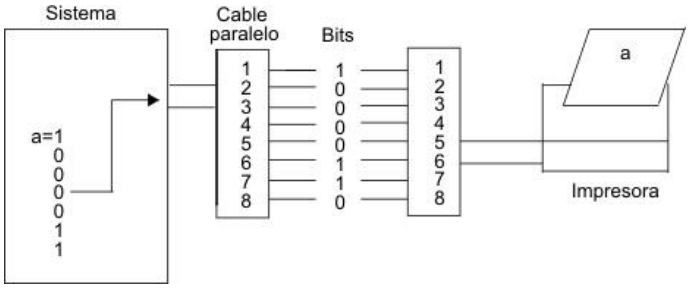


Fig. 6.2. Ejemplo de comunicación paralela[11]

Para ambos tipos de comunicación, la generación y la recepción de datos en la transmisión se utiliza una señal de reloj. La señal de reloj es una señal de '0s' y '1s' que se repite con cierta frecuencia pudiendo estar más tiempo como bit '1' que como bit '0' o viceversa (aperiódica).[12]

La ventaja de una sobre la otra es que la comunicación serie necesita menos canales o conexiones que una comunicación paralela ante igual cantidad de información. Esto supone que los cables utilizados para la conexión son más baratos que los que se utilizan en paralelo. [12]

Sin embargo, si ambas transmisiones se realizan a la vez en el tiempo, la comunicación paralela es más eficiente por lo que necesita menos velocidad que la comunicación serie. Asimismo, aunque la comunicación paralela es más eficiente, tiene problemas de sincronización lo que supone un lastre en el rendimiento.

Por otro lado, la comunicación en serie es más segura ante posibles errores ya que es una única línea de transmisión donde los bits se disponen de manera secuencial. En cambio, la comunicación paralela ser envían los datos juntos lo que la hace más sensible al ruido y como consecuencia más propensa a errores.

En resumen, en la comunicación serie es más robusta ante errores, más lenta y sencilla, en cambio, la comunicación paralela es menos fiables, rápida y más costosa. Esto significa que a distancias más cortas es más eficiente la transmisión paralela que la transmisión serie que es mejor en largas distancias.[12]

La conclusión que se saca de este análisis es que la comunicación paralela, aunque sea la más rápida y eficiente, esto no es una prioridad, debido a que tiene más peso su elevado coste y su sensibilidad a interferencias como las que puede causar la radiación en los cables que van conectados a las Zybo. Por lo tanto, tanto estas últimas como la Raspberry Pi se utilizará la comunicación serial.

Por último, cabe mencionar una serie de motivos adicionales que da más fuerza a la decisión de utilizar esta comunicación en detrimento de la paralela, estos son el soporte nativo de la Raspberry Pi en la comunicación serial y la posibilidad de compensar la ineficiencia de la comunicación serie, aumentando la frecuencia.

Por otro lado, la comunicación serie maneja tres tipos de transmisión:

- **Full duplex**. Puede recibir y enviar datos indistintamente.

- **Duplex o half-duplex**. Puede recibir y transmitir datos, pero no las dos cosas a la vez.
- **Simplex**. Solo puede recibir o transmitir.

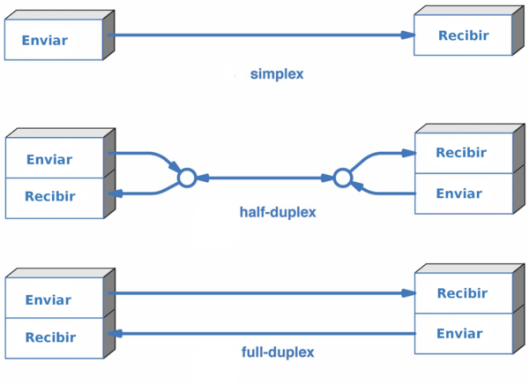


Fig. 6.3. Tipos de transmisión

En este caso, lo más útil es implementar una comunicación serie que sea full-duplex, para una comunicación más rápida.

Por otra parte, para la implementación de la comunicación serie en los dispositivos, se requieren una serie de protocolos, los cuales se dividen en dos categorías según su modo de transmisión:

- **Asíncrona**. La comunicación asíncrona es aquella que no precisa de una coordinación temporal entre el emisor y el receptor, es decir, los relojes de cada extremo de la comunicación no están sincronizados ni tienen porque funcionar a la misma frecuencia. La forma que tienen de coordinar sus relojes es mediante un bit de 'start' y 'stop'. Ejemplos de este tipo de es el protocolo UART.
- **Síncrona**. La comunicación síncrona se basa en el envío de información de manera coordinada a través de una señal de reloj a cierta frecuencia compartida por ambos extremos. Esto provoca una sincronización temporal en la comunicación en ambos extremos. Este tipo de transmisión es más robusta frente a errores lo que permite utilizar mayor velocidad en el envío y recepción de datos. Un ejemplo de este tipo de comunicación son los protocolos I2C y SPI.

Como se ha mencionado en su apartado los protocolos serial soportados nativamente por la Raspberry Pi son: UART, SPI e I2C. Por lo tanto, estos son los que se van a implementar para el intercambio de información.

6.3 SPI

El protocolo SPI es un protocolo serie síncrono que transmite los datos en modo full-duplex. Este permite el intercambio entre solamente dos dispositivos al mismo tiempo, con un comportamiento maestro-esclavo.[13]

En este protocolo se define un dispositivo maestro que es el encargado de controlar el bus para el intercambio de información y genera una señal de reloj que compartirá con el otro dispositivo, el esclavo. Tanto el maestro como el esclavo pueden tanto enviar como recibir datos. Esta información se almacena en un tipo de memoria denominada registro de

desplazamiento, cuyos datos almacenados se desplaza en cada ciclo de reloj, la muestra de ello se representa en la **figura**.[13]

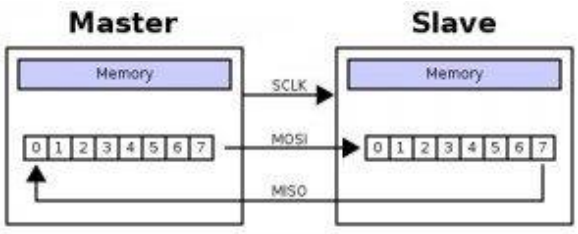


Fig. 6.4. Bus SPI[13]

El protocolo SPI cuenta con un bus que está formado por cuatro líneas lógicas[13]:

- SCLK (SPI CLOCK). Línea de salida del maestro que envía la señal de reloj a los esclavos para sincronizarlos.
- MOSI (Master Output - Slave Input). Línea de salida del maestro que utiliza para transmitir información hacia el esclavo
- MISO (Master Input - Slave Output). Línea de entrada del maestro por donde recibe la información que proviene del esclavo.
- SS (Slave Select). Línea de salida que controla el maestro para seleccionar, y por lo tanto permitir la comunicación con un eslavo. Normalmente, el nivel bajo habilita y el nivel alto deshabilita, que es su estado por defecto.
- GND (Ground). Es la línea que representa masa.

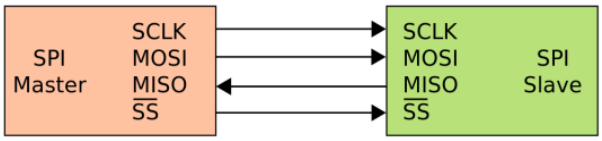


Fig. 6.5. Líneas que conforman el bus SPI[14]

Lo que en el maestro son líneas de salida en el eslavo son de entrada, y viceversa.

Por otro lado, en el protocolo SPI se definen cuatro modos de funcionamiento. Estos se basan en las condiciones iniciales (CPOL: polaridad) y comportamiento (CPHA: fase) de la señal de reloj. Estos modos de funcionamiento son[13]:

- Modo 0 (CPOL = 0, CPHA=0). En este modo la señal de reloj comienza en estado bajo (0) y la información se envía en cada transición de bajo a alto.
- Modo 1 (CPOL = 0, CPHA=1). En este modo la señal de reloj comienza en estado bajo (0) y la información se envía en cada transición de alto a bajo.
- Modo 2 (CPOL = 1, CPHA=0). En este modo la señal de reloj comienza en estado alto (0) y la información se envía en cada transición de bajo a alto.
- Modo 3 (CPOL = 1, CPHA=1). En este modo la señal de reloj comienza en estado alto (0) y la información se envía en cada transición de alto a bajo.

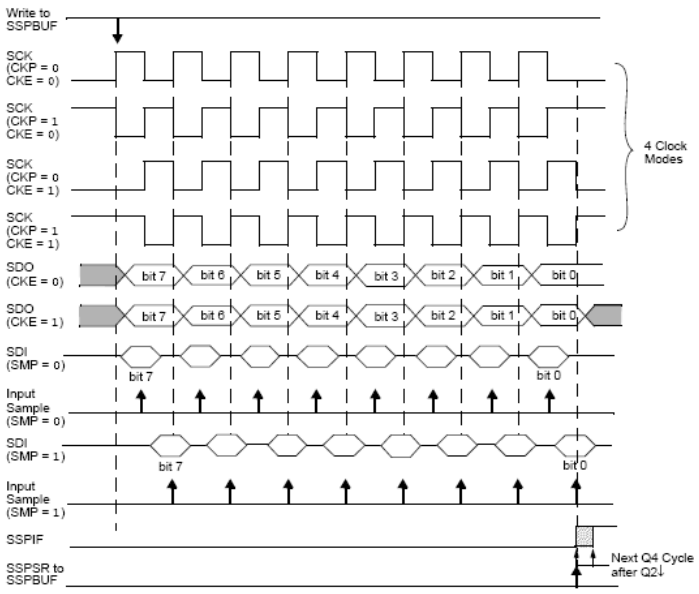


Fig. 6.6. Diferentes modos SPI[15]

Otro parámetro que se puede configurar en la comunicación es la transferencia de datos desde MSB o desde LSB, es decir, si es una información de 8 bits que se desea enviar primero el bit 0(LSB) o primero el bit 7(MSB).[13]

Esta configuración es independiente para cada esclavo, pero es ineficiente que el maestro tenga que cambiar de configuración para adaptarse cada vez que se comunica con un esclavo diferente.[16]

En este protocolo se puede definir y conectar más de un esclavo debido su comunicación a través de un bus. Sin embargo, solo puede definirse un solo maestro. Existen dos formas de conectar los esclavos al maestro[16]:

- **Paralela**. En esta forma hay una sola línea SCLK, MOSI y MISO, donde se conectan cada uno de los esclavos de forma paralela. Y una línea SS del maestro por cada esclavo disponible.
  Esta forma es más robusta ante errores, ya que si un dispositivo falla el resto sigue funcionando. Por el contrario, se necesita más de una línea por esclavo y no siempre están implementadas.
- **Cascada (Daisy Chain)**. En esta forma la línea MOSI del maestro se conecta al que se define como primer esclavo, y el resto de los esclavos se conecta con el MISO del anterior. Existe una sola línea SCLK donde se conectan todos los esclavos y una sola línea SS donde se conectan estos de forma paralela. Esto supone que el maestro envía los datos al primer esclavo y recibe la respuesta desde el ultimo.
  A diferencia de la forma anterior es poco robusta ante errores, si un dispositivo falla la comunicación entera se cae, y, además proporciona una mala propagación de errores, ya que las líneas de datos MOSI y MISO entre esclavos se conectan en serie. Sin embargo, solo se necesita una sola línea de selección para un número indeterminado de esclavos.

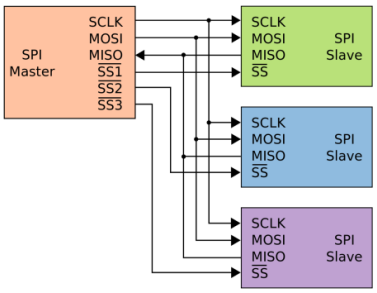


Fig. 6.7. Bus SPI conectado de forma paralela

Figura. Bus SPI conectado de forma paralela[16]

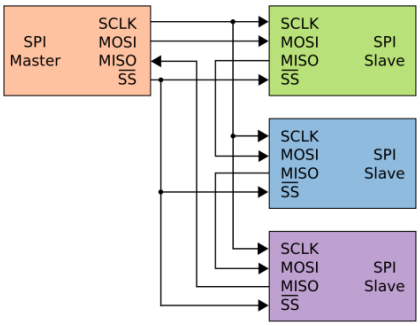


Fig. 6.8. Bus SPI conectado en forma de cascada

Figura. Bus SPI conectado en forma de cascada[16]

Imaginemos una transferencia de información utilizando el protocolo SPI en modo 0 y conectados en forma paralela (una línea SS por cada esclavo). Cada bit que transmite el maestro al esclavo, este lo devuelve de nuevo al maestro. La transferencia se realizará en ambos lados desde el MSB a LSB.

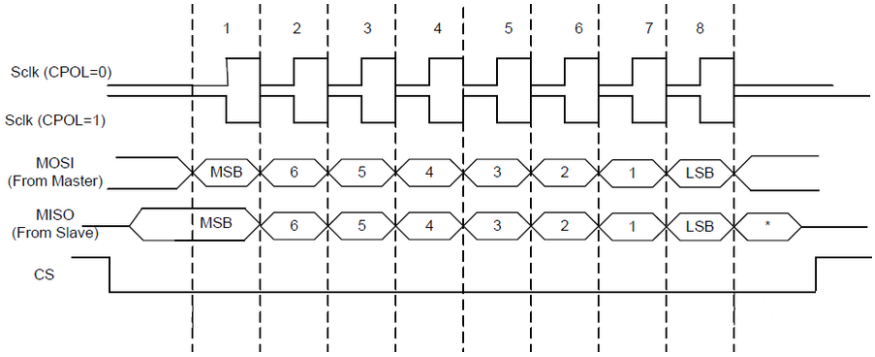


Fig. 6.9. Funcionamiento del protocolo SPI (Modo 1) [17]

Las principales ventajas del protocolo SPI con respecto a otros son[14], [18]:

- Su alta velocidad de transmisión.
- Debido a que funciona en modo full-duplex las líneas de transmisión entre maestro y esclavo está separadas (MOSI y MISO).

- La transmisión no se divide y no se producen interrupciones, ya que no hay bits que indiquen el inicio y el final de la comunicación.

Por el contrario, sufre la siguiente serie de desventajas[14], [18]:

- Utiliza 3 cables, y un cable SS adicional por cada esclavo que se conecta. Por lo que por cada vez se requiere conectar un esclavo se puede necesitar una modificación de hardware. Esto supone un coste alto.
- No hay ningún tipo de mecanismo que permita saber si el mensaje se ha recibido y si se ha recibido correctamente.
- La longitud de los mensajes debe ser conocida por ambos dispositivos.
- Solo funciona en distancias cortas, alrededor de medio metro.

6.3.1 Diseño del programa para la comunicación mediante el protocolo SPI

El siguiente paso es el diseño del programa que se encarga de utilizar el protocolo SPI para el intercambio de información entre la Raspberry Pi y las Zybo.

6.3.1.1 SPI en la Raspberry Pi

Como se ha comentado en su apartado, la Raspberry Pi tiene implementado dos interfaces SPI, SPI0 con dos líneas SS y SPI1 otra con tres líneas SS. Sin embargo, en este caso nos vamos a centrar en la interfaz SPI0, ya que la otra tiene ciertas limitaciones en velocidad.

La Raspberry Pi posee 5 pines reservados para la comunicación I2C: CE0, CE1, MOSI, MISO y SCLK. Por consecuencia, se utilizan los pines asignados 24, 26, 19, 21 y 23 que se muestran en la **figura**.

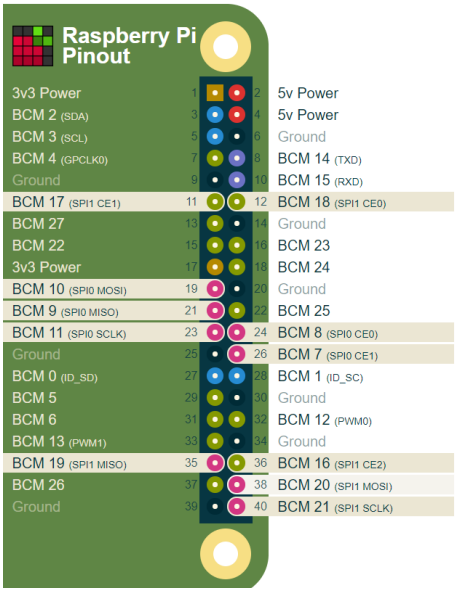


Fig. Pines reservados para el bus SPI[19]

El controlador SPI que se incluye permite que la Raspberry solo pueda definirse como maestro. En cuanto a la velocidad de la señal de reloj en este protocolo, soporta las siguientes frecuencias:

TABLA 6.1. FRECUENCIAS SOPORTADAS POR EL MÓDULO SPI EN LA RASPBERRY PI[20]

Divisor de señal de reloj	Velocidad/Frecuencia (MHz)
2	125.0MHz
4	62.5MHz
8	31.2MHz
16	15.6MHz
32	7.8MHz
64	3.9MHz
128	1953kHz
256	976kHz
512	488kHz
1024	244kHz
2048	122kHz
4096	61kHz
8192	30.5kHz
16384	15.2kHz
32768	7629Hz

Por lo tanto, esta es la encargada de gestionar la comunicación mediante este protocolo. Para poder utilizar el bus SPI es necesario disponer del software adecuado, para ello se requieren tener instaladas una de las siguientes librerías:

- **WiringPi**. Esta librería se puede utilizar en dos lenguajes de programación diferentes, en C y en Python. Esta librería permite el acceso a los pines GPIO, al módulo I2C, al módulo SPI, entre otras funciones. En cuanto al módulo SPI tiene solo dos métodos[21]:
    - **int wiringPiSPISetup (int channel, int speed)**. Esta función configura el protocolo SPI. En ‘channel’ se indica la línea SS que se va a utilizar 0 o 1, y en ‘speed’ se indica la frecuencia de la señal de reloj representada en Hz, esta es número entero de un rango que va desde los 500.000 a los 32.000.000. Si la función devuelve un -1 significa que la configuración ha fallado.
    - **int wiringPiSPIDataRW (int channel, unsigned char *data, int len)**. Esta función realiza una lectura y escritura de manera simultánea. En ‘channel’ se indica el mismo valor que se ha indicado en la configuración del SPI, en ‘data’ se indica el buffer de los datos que se van a enviar al

esclavo y posteriormente se sustituyen con lo recibido de este y, por último, en 'len' se indica la longitud del buffer, es decir, la cantidad de bytes que se van a enviar.

- **Bcm2835**. Esta librería esta compilada en C por lo que solo se puede utilizar en este lenguaje de programación. Como en la librería anterior, esta también permite el acceso a los pines GPIO, al módulo I2C, al módulo SPI, entre otras funciones. En lo que concierne al módulo SPI, los métodos implementados más importantes son[22]:
    - **int bcm2835_spi_begin ()**. Inicializa los pines GPIO dedicados a SPI de la Raspberry Pi. Devuelve un valor 1 si todo ha ido bien, si no devuelve un 0.
    - **void bcm2835_spi_setBitOrder (uint8_t order)**. Configura el orden en el que se transmite la información, MSB o LSB.
    - **void bcm2835_spi_setClockDivider (uint16_t divider)**. Configura el divisor de la señal de reloj que determina la frecuencia del reloj SPI.
    - **void bcm2835_spi_setDataMode (uint8_t mode)**. Configura el modo de funcionamiento del SPI.
    - **void bcm2835_spi_chipSelect (uint8_t cs)**. Selecciona la línea SS a utilizar en la comunicación, en el parámetro 'cs' se indica el valor 0 o 1. Esta se selecciona en el inicio de la transmisión.
    - **void bcm2835_spi_transfernb (char *tbuf, char *rbuf, uint32_t len)**. Inicia la transferencia entre el maestro y el esclavo. El parámetro 'cbuf' indica el buffer con los datos que se quiere transmitir desde el maestro, el parámetro 'rbuf' indica el buffer donde se reciben los datos que provienen del esclavo y, por último, el parámetro 'len' indica el tamaño de cada uno de los datos que se va a transmitir. Por lo tanto, todos los datos deben tener el mismo tamaño.
    - **void bcm2835_spi_end ()**. Finaliza las operaciones SPI y libera los pines dedicados.

De las dos librerías presentadas, es escoge esta última para implementar la comunicación SPI en el programa debido a que es mucho más configurable de acuerdo con las necesidades. Por lo tanto, el lenguaje de programación elegido para desarrollar el programa es C.

A la hora del desarrollo se plantea un programa que sea capaz de enviar 9 números de un rango del 1 al 9 y que estos sean devueltos por el esclavo a través del bus SPI. Durante la transmisión se debe mostrar en pantalla los datos enviados y recibidos. Tras la recepción de los datos por parte de este último se debe comprobar si coincide con la información, si es correcta tiene que aparecer un mensaje verde en la ventana de comandos y si no, aparecer en rojo. Además, debido a que disponemos de dos líneas SS, podemos conectar dos esclavos.

Antes de la ejecución del programa se debe configurar la línea asociada al esclavo con el que queremos intercambiar la información. La velocidad del bus SPI es suficiente con

establecerla en un divisor de 128, es decir, en torno a los 2MHz**,** ya que más allá de esta velocidad la señal no es estable es mucho más vulnerable al ruido. El cálculo se muestra en la siguiente ecuación.

$$Velocidad\ SPI = \frac{Reloj\ Raspberry\ Pi}{Divisor\ de\ reloj} = \frac{250MHz}{128} = 2MHz \qquad (6.1)$$

El modo de funcionamiento del bus SPI es el 0 (CPOL=0, CPHASE=0).

El diagrama de flujo que describe el funcionamiento del programa se muestra en la siguiente figura.

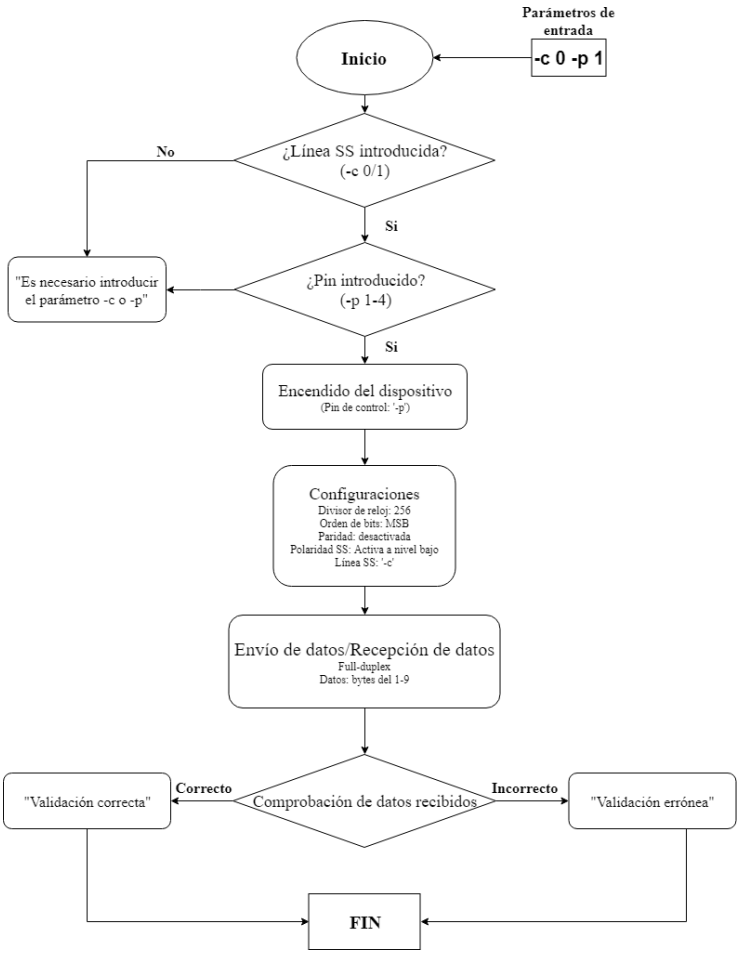


Fig. 6.10. Diagrama de flujo de la interfaz SPI

Antes de nada, para poder utilizar la librería bcm2385 es necesario añadir la siguiente línea al código:

```
#include <bcm2835>
```

Después, se implementan los métodos enunciados anteriormente de acuerdo con el flujo que se define en la figura anterior. El programa se implementa en el archivo spi.c a través del editor 'nano' de la interfaz de línea de comandos que ofrece PuTTY.

En conclusión, con este programa podemos probar de manera el funcionamiento del protocolo SPI para el intercambio de información.

El código completo se encuentra en el Anexo M.

6.3.1.2 SPI en la Zybo

En la parte de la Zybo, implementaremos una interfaz SPI que trabajará como esclavo. Para ellos, nos apoyaremos en uno de los lenguajes de programación estándar de las FPGA, VHDL. La herramienta que se ha utilizado para el desarrollo del módulo SPI ha sido Vivado 2015.2.

Teniendo en cuenta el programa realizado en la Raspberry Pi, se deben plantear unos requisitos que permitan que ambas se comuniquen. Los requisitos planteados son los siguientes:

- Muestreo del bus SPI usando la señal de reloj de la Zybo.
- Devolución a máster de cada bit recibido por este.
- Longitud configurable del tamaño de los datos recibidos.
- Implementación como mínimo del funcionamiento modo 0.
- Envío de información como MSB (bit más significativo).
- Señal RESET que reinicie los registros de la Zybo.

En primer lugar, se define una entidad en VHDL donde se definen las entradas y salidas requeridas para un dispositivo esclavo SPI. Además, siguiendo el tercer requisito se define una longitud de datos genérica mediante el parámetro 'width' (por defecto es 8) y el parámetro 'width_counter' que es el exponente de la longitud de datos en base 2 (por defecto es 3). Estos parámetros permiten cambiar su configuración al instanciarla como componente. En la siguiente figura se muestra el código de la entidad implementada.

```
34 entity SPI is
35 generic(width: INTEGER := 8;
36 width_counter: INTEGER := 3);
37     Port ( CLK : in STD_LOGIC;
38            RST : in STD_LOGIC;
39            SCLK : in STD_LOGIC;
40            MOSI : in STD_LOGIC;
41            SS_N : in STD_LOGIC;
42            MISO : out STD_LOGIC);
43 end SPI;
```

Fig. 6.11. Entradas y salidas SPI del esclavo en VHDL

Las entradas y salidas que se observan en la figura se asignan varios pines de puerto pmod a través de un archivo XDC. La asignación es la siguiente:

- U17: MISO.
- B13: MOSI.
- T17: SCLK.
- Y17: SS.

Por otro lado, el primer requisito es necesario debido a que normalmente el bus SPI es más lento que la señal de reloj de una FPGA. En este caso el reloj de la Zybo tiene una frecuencia de entorno a los 50MHz y la comunicación se realiza a 2MHz.

Por lo tanto, sino se muestrea el bus SPI el esclavo, este leería el mismo bit varias veces antes de que el maestro enviase el siguiente, lo que supondría una transmisión errónea. En la siguiente figura se representa el muestreo de dos señales de reloj, donde CLK funciona 16 veces más rápida que SCLK.

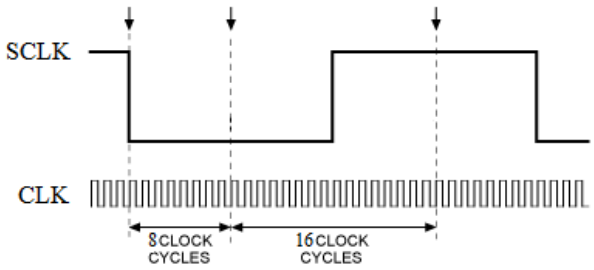


Fig. 6.12. Muestreo de la señal de reloj SPI[23]

Como se ha explicado en el párrafo anterior, cuando la señal CLK ha cumplido 8 ciclos, ha leído 8 veces el mismo valor, y no es hasta 16 ciclos después cuando esté valor cambia.

Para muestrear el bus lo que se hace es pasar la señal SCLK como entrada de un Flip-Flop y como señal de sincronismo el reloj de la FPGA, CLK.

Al igual que con la señal de reloj de SPI, se muestrean de la misma forma las otras dos líneas MISO, MOSI y SS.

Sin embargo, con esta implementación surge un problema con la señal a la salida de estos Flip-Flops, la señal está deformada. A este problema se le conoce como meta estabilidad. En la siguiente figura se muestra de forma gráfica una señal con meta estabilidad.

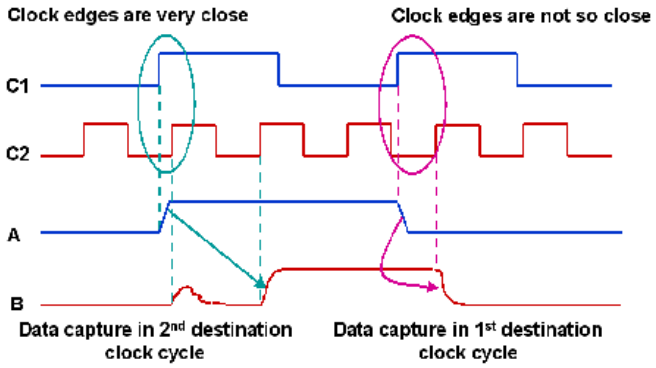


Fig. 6.13. Metaestabilidad de una señal[24]

Para solventar la metaestabilidad se introduce a las señales SCLK y SS anteriores a través de un registro de desplazamiento de tres bits, y las señales MISO y MOSI en un registro de desplazamiento de dos bits.

Por otro lado, se implementan una serie de detectores de flanco para las señales de entrada que se necesitan como condición para el envío y recibimiento de bytes, estas son la señal de reloj SCLK y la señal de selector de esclavo SS. Es por este motiva por

el cual se introducen estas señales a través de un registro de desplazamiento de tres bits y no de dos.

Una vez que la señales se pueden utilizar en la rutina de la comunicación. Se implementan dos registros de desplazamiento, uno encargado de recibir los bits del maestro y otro que envía esto bits, siempre empezando por el bit MSB. En cada flanco de subida del reloj, siempre y cuando la señal SS esté a nivel bajo, se inicia la transferencia entre maestro y esclavo. El único modo compatible con esta interfaz es el modo 0.

El motivo de esta implementación surge porque la comunicación es full-duplex. Por lo tanto, en la interfaz SPI no tiene sentido utilizar máquinas de estado cuando la información se recibe y envía de manera simultánea.

Cabe destacar, que para el reinicio de las rutinas se ha implementado una señal de reset que reinicio el estado de los registros.

El flujo de la interfaz SPI implementada en la Zybo se muestra en la siguiente figura.

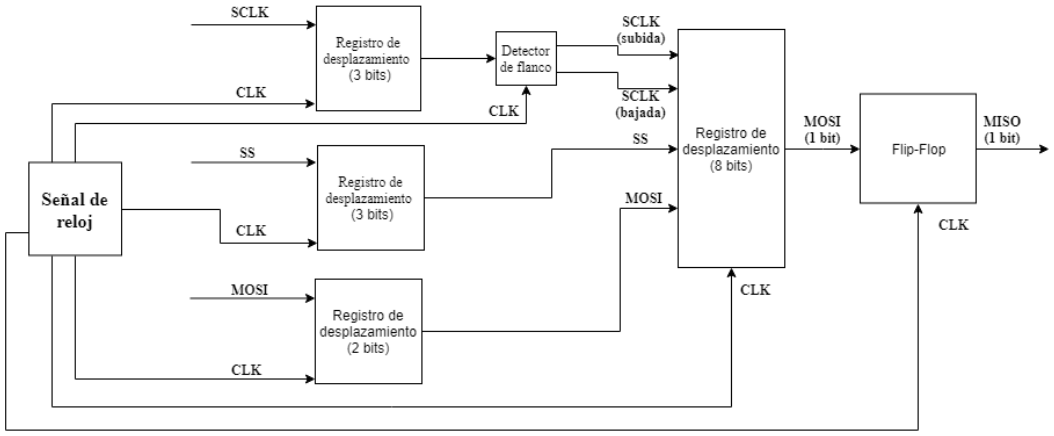


Fig. 6.14. Diagrama de flujo de la interfaz SPI en la Zybo

En lo que respecta, a la programación de la interfaz SPI en la Zybo, se debe seguir el procedimiento que se incluye en el Anexo.

Finalmente, con la interfaz SPI implementada en la Zybo se puede llevar a cabo un intercambio de información a través del protocolo SPI. Las pruebas de esta comunicación se realizarán en capítulos posteriores.

El código completo se encuentra en el Anexo N.

6.4 I2C

El protocolo i2c es un protocolo serie síncrono de transmisión bidireccional que necesita únicamente dos líneas para su funcionamiento y una tercera de referencia[25]:

- SDA. Línea para la transmisión de los datos.
- SCL. Línea para la señal de reloj que se encarga de controlar el maestro.
- GND. Es la línea que representa masa.

Este protocolo posee una arquitectura maestro-esclavo, donde el maestro es el que gestiona la comunicación. El maestro es el único que puede iniciar la comunicación, además de que los esclavos no pueden hablar directamente con ellos, primero deben pasar por el maestro.[26]

En la comunicación, todos los esclavos se conectan al mismo bus donde cada uno de ellos tiene asociada una dirección única y propia que viene determinada por hardware o por software.

El bus I2C es de tipo drenaje abierto, por lo que se debe polarizar en nivel alto utilizando resistencias pull-up con las líneas de alimentación Vcc. Esto permite que se conecten en paralelo múltiples entradas y salidas.[27]

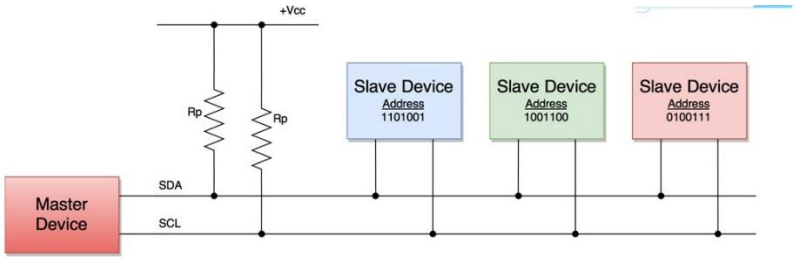


Fig. 6.15. Bus I2C con el maestro y tres esclavos conectados[28]

El protocolo I2C posee 4 modos de funcionamiento, en cada modo la velocidad es mayor:

- Modo estándar. Soporta una velocidad máxima de 100kbps.
- Modo rápido. Soporta una velocidad máxima de 400kbps.
- Modo alta velocidad. Soporta una velocidad máxima de 3.4Mbps.
- Modo ultra rápido. Soporta una velocidad máxima de 5Mbps.

El funcionamiento de I2C se basa en el intercambio de mensajes entre maestro y esclavos. Cada uno de estos mensajes están divididos en tramas de donde se distingue un bit de inicio de la comunicación, la dirección del esclavo al que se quiere enviar el mensaje, bits de control y uno o más tramas de datos[29]. En la **figura** se muestra el formato de un mensaje en I2C.

Fig. 6.16. Tramas de un mensaje I2C[29]

Las tramas del mensaje se producen en las siguientes situaciones:

- **Inicio**. En el inicio de la comunicación, la secuencia de inicio que se muestra en la **figura** se produce cuando la línea SDA pasa de nivel alto a nivel bajo y a su vez la línea SCL se mantiene en nivel alto.
- **Final**. Para finalizar la comunicación con un esclavo la línea SDA cambia de nivel bajo a nivel alto y a su vez la línea SCL cambia de nivel bajo a alto.
- **Dirección del esclavo**. Es una serie de bits únicos e irrepetibles en el bus, con un tamaño estándar de 7 bits. En este sentido, existe una ampliación de este tamaño a los 10 bits por si se necesitaran más número de direcciones. Esta trama la recibe cada esclavo, y se identifica como el dispositivo con el que debe comunicarse el maestro. Por lo tanto, teóricamente el número máximo de esclavos que se podrían conectar es de 128 dispositivos. Sin embargo, en realidad solo es posible conectar 112 ya que las 16 direcciones restantes están reservadas para el funcionamiento interno del protocolo. Estas direcciones especiales se muestran en la **Figura**.
- **Bit de lectura/escritura**. Es un bit de control que envía al esclavo para indicarle si debe escribir en sus registros la información que envía el maestro (Escritura = SDA a nivel bajo), o si el necesita que le envíen la información que contenga o pueda enviar el esclavo (Lectura = SDA a nivel alto).
- **Bit ACK/NACK**. Tras casa bloque de datos enviado en un menaje, se incluye otro bit de control denominado acknowledge/no-acknowledge (reconocido/no-reconocido). Este bit de control tiene la función de informar si el mensaje se ha enviado correctamente. Si el mensaje ha llegado con éxito, se retorna un bit ACK (línea SDA a nivel bajo) al que lo ha enviado. Sin embargo, si el mensaje no ha llegado o se han producido errores por el caminito, se retorna un bit NACK con la línea SDA a nivel alto.[29]

SLAVE ADDRESS	$\overline{R/W}$ BIT	DESCRIPTION
0000 000	0	General call address
0000 000	1	START byte
0000 001	X	CBUS address
0000 010	X	Reserved for different bus format
0000 011	X	Reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	X	Reserved for future purposes
1111 0XX	X	10-bit slave addressing

Fig. 6.17. Direcciones reservadas protocolo I2C[30]

Por último, cabe destacar las fortalezas de I2C frente a otros protocolos serie:

- Solo se requiere de dos cables para su funcionamiento, SDA y SCL (sin contar la masa).
- Posee herramientas para asegurar la llegada y la recepción de la información (bits ACK).
- Soporta una gran cantidad de dispositivos esclavos conectados.
- Facilidad para la conexión de nuevos dispositivos sin modificaciones de hardware.

Sin embargo, también dispones de una serie de desventajas que pueden pesar a la hora de utilizarlo en ciertas situaciones. Estas son:

- No trabaja en full-duplex.
- El hardware es más complejo que, por ejemplo, SPI.
- No hay control de errores.
- Es más lento que SPI.
- Longitud de los datos limitada a 8 bits por trama.[29]

6.4.1 Diseño de la comunicación mediante el protocolo I2C

De la misma manera que en el apartado de la SPI, se requiere implementar un programa tanto en la Raspberry Pi como en la Zybo para que ambas partes se comuniquen a través del protocolo I2C.

Para ello, se divide en dos partes, una donde se explica la implementación de la interfaz en la Raspberry Pi y donde se implementa en la Zybo. En cada una de ella se explica el lenguaje utilizado para el desarrollo, las librerías para la implementación de la interfaz de comunicación, el funcionamiento diseñado y los ajustes necesario para la comunicación del programa, su compilación y su ejecución.

6.4.1.1 I2C en Zybo

En el otro extremo de la comunicación I2C se encuentra la Zybo. A diferencia de la Raspberry Pi, la Zybo no implementa ningún tipo de hardware ni software que permita comunicarse con ella por I2C.

En este sentido, se implementa una interfaz I2C para completar el otro extremo de la comunicación I2C del proyecto. Para el desarrollo y la programación de dicha interfaz se recurre a la herramienta utilizada anteriormente, Vivado 2015.2. En la que se utiliza un lenguaje estándar para programar FPGA, VHDL.

En esta sección se la Zybo, implementamos una interfaz SPI que posee la función de esclavo. Para ello, nos apoyaremos en uno de los lenguajes de programación estándar de las FPGA, VHDL. La herramienta que se ha utilizado para el desarrollo del módulo I2C ha sido Vivado 2015.2.

Teniendo en cuenta el programa realizado en la Raspberry Pi, se deben plantear unos requisitos que permitan que ambas se comuniquen. Los requisitos se implementan son los siguientes:

- Muestreo del bus I2C usando la señal de reloj de la Zybo.
- Devolución al maestro de cada bit recibido por este.
- Longitud de máxima de 8 bits de los datos recibidos.
- Envío de información como MSB (bit más significativo).
- Configuración de la dirección I2C mediante sus 'switches' e indicada a través de sus leds.
- Señal RESET que reinicie los registros de la Zybo.

En primer lugar, se define una entidad en VHDL donde se definen las entradas y salidas requeridas para una interfaz I2C. En la siguiente figura se muestra el código de la entidad implementada.

```
 9 entity I2C is
10     port (
11     SCL             : inout STD_LOGIC;
12     SDA             : inout STD_LOGIC;
13     CLK             : in STD_LOGIC;
14     SWT             : in STD_LOGIC_VECTOR(3 DOWNTO 0);
15     LED             : out STD_LOGIC_VECTOR(3 DOWNTO 0));
16 end I2C;
```

Fig. 6.18. Entradas y salidas interfaz I2C

En este caso ambas señales son salida y entrada a la vez, ambos pueden establecer valores a la línea. Aunque en el uso habitual, solo el maestro modifica la línea SCL.

Las entradas y salidas que se observan en la figura se asignan varios pines de puerto pmod, los cuatro 'switches' y los cuatro leds. a través de un archivo XDC. La asignación es la siguiente:

- V12: SCL
- W16: SDA
- M14, M15, G14, D18: Leds.
- G15, P15, W13, T16: Switches.

Cómo sucedía en el aparatado SPI, es necesario muestrear las señales SDA y SCL a través de la señal de reloj que posee la Zybo. Además es muy importante tener en cuenta que sino muestreamos adecuadamente las señales pueden producirse problemas de metaestabilidad en ellas.

Con el propósito en evitar la metaestabilidad se utilizan dos registros de desplazamiento de tres bits cuyas salidas van conectadas a detectores de flanco de subida y de flanco de bajada.

De acuerdo al funcionamiento en half-duplex del protocolo I2C se decide implementar una máquina de estados. La siguiente figura se muestra los estados de esta máquina.

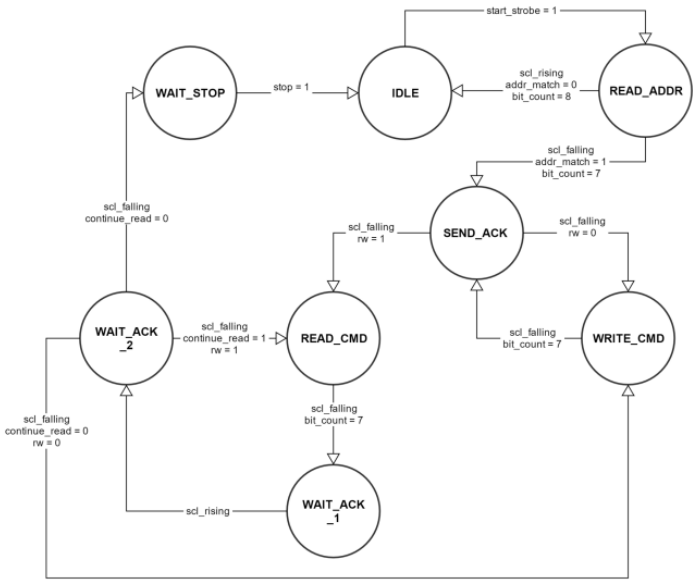


Fig. 6.19. Máquina de estados en la interfaz I2C

Por otro lado, se configuran los registros de recepción para almacenar un máximo de 8 bits de acuerdo a los requisitos definidos.

Cómo se ha explicado en la introducción del protocolo I2C, cada esclavo tiene asignada una dirección única que permite su distinción a la hora de comunicarse con el maestro. Para hacer más versátil la interfaz I2C se decide implementar que la dirección I2C del esclavo sea configurable a través de los 'switches' que contiene la Zybo. Cada uno de estos interruptores representan un bit de la dirección de 7 bits. En concreto, los bits del del seis al tres. Esto se debe a que existen una serie de direcciones reservadas para el funcionamiento interno del protocolo I2C.

A continuación se muestra una figura con el flujo completo de la interfaz I2C.

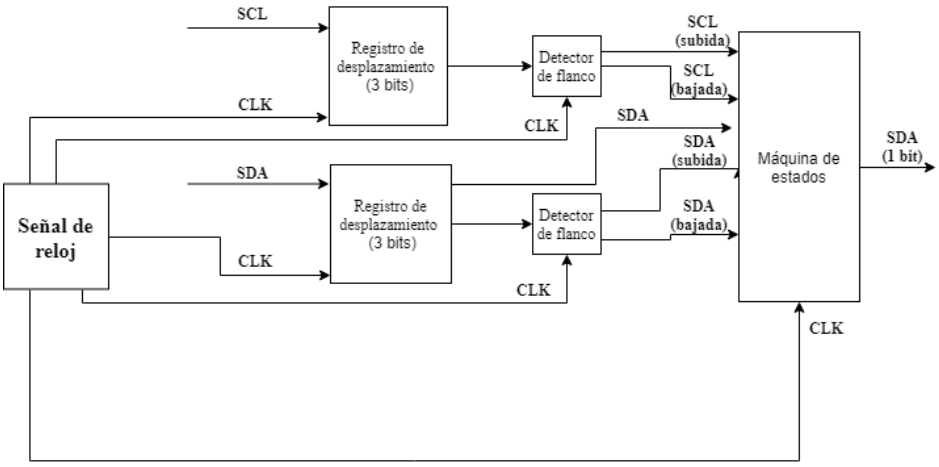


Fig. 6.20. Diagrama de flujo de la interfaz I2C

En lo que respecta, a la programación de la interfaz I2C en la Zybo, se debe seguir el procedimiento que se incluye en el Anexo L.

En conclusión, con esta parte implementada de la comunicación I2C se está más cerca de realizar transferencia de datos a través de este protocolo.

El código completo se encuentra en el Anexo P.

6.4.1.2 I2C en Raspberry Pi

Otro de los protocolos serie implementado en la Raspberry Pi es el protocolo I2C. Como se muestra en el apartado GPIO, la Raspberry Pi posee 4 pines reservados para la comunicación: SDA, SCL, EEPROM SDA y EEPROM SCL.

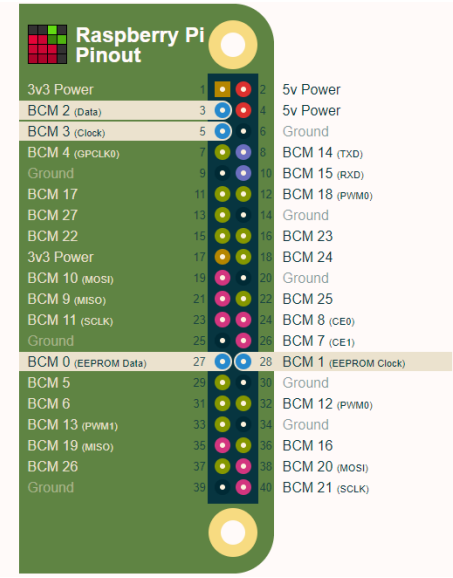


Fig. 6.21. Pines reservados para la comunicación I2C[7]

En este caso, nos quedaremos con los pines 3 y 5(SDA y SCL). Ya que los 27 y 28 están destinados a comunicarse con una EEPROM.

No es necesario añadir resistencias pull-up para las conexiones con las líneas SDA y SCL, ya vienen implementadas por defecto en la Raspberry Pi.

Antes de empezar a desarrollar debe comprobar que puerto I2C está disponible en la Raspberry Pi. Para ello, como hemos visto anteriormente, a través de la interfaz de comandos que nos proporciona PuTTY introducimos el comando que lista los dispositivos en la ruta /dev que filtrándolos por la palabra ‘i2c’:

```
ls /dev/*i2c*
```

Si el puerto I2C está disponible, los modelos de Raspberry Pi que se utilizan en este proyecto deben responder con la siguiente línea:

```
/dev/i2c-1
```

Cabe destacar, que la interfaz I2C que se implementa en la Raspberry Pi solo actúa como maestro, por lo que será la encargada de controlar la comunicación. La velocidad por defecto de la interfaz está configurada a 100kbps.

De igual modo que en la sección SPI, para implementar un programa para la Raspberry Pi que utilice el bus I2C, se debe disponen de al menos una de estas dos librerías:

- **WiringPi**. Esta librería se puede utilizar en dos lenguajes de programación diferentes, en C y en Python. Esta librería permite el acceso a los pines GPIO, al módulo I2C, al módulo SPI, entre otras funciones . En cuanto al módulo I2C, tiene solo dos métodos[31]:
    - **int wiringPiI2CSetup (int devId)**. Inicializa el dispositivo I2C. En el parámetro 'devId' debe introducirse la dirección del esclavo con el que se quiere iniciar la comunicación. La función devuelve un descriptor de archivo si todo ha ido bien, o -1 si ha habido algún error.
    - **int wiringPiI2CRead (int fd)**. Lee del dispositivo esclavo. En el parámetro 'fd' se introduce el descriptor de archivo devuelto por la función wiringPiI2CSetup.
    - **int wiringPiI2CWrite (int fd, int data)**. Escribe en el dispositivo esclavo. En el parámetro 'fd' se introduce el descriptor de archivo devuelto por la función wiringPiI2CSetup, y en el parámetro 'data' se indica el buffer de los datos que se van a enviar al esclavo.
- **Bcm2835**. Esta librería esta compilada en C por lo que solo se puede utilizar en este lenguaje de programación. Como en la librería anterior, esta también permite el acceso a los pines GPIO, al módulo I2C, al módulo SPI, entre otras funciones. En lo que concierne al módulo I2C, los métodos implementados más importantes son[32]:
    - **int bcm2835_i2c_begin ()**. Inicializa los pines GPIO dedicados a I2C de la Raspberry. Si la función devuelve un 1 se ha inicializado correctamente, si no devuelve un 0.
    - **void bcm2835_i2c_end ()**. Finaliza las operaciones I2C y libera los pines dedicados.
    - **void bcm2835_i2c_setSlaveAddress (uint8_t addr)**. Configura la dirección del esclavo con el que se quiere iniciar la comunicación.
    - **void bcm2835_i2c_setClockDivider (uint16_t divider)**. Configura el divisor de la señal de reloj que determina la frecuencia del reloj I2C.
    - **void bcm2835_i2c_set_baudrate (uint32_t baudrate)**. Configura la velocidad de la transmisión del bus I2C (por defecto, 100kbps).
    - **uint8_t bcm2835_i2c_write (const char *buf, uint32_t len)**. Lee una serie de datos del esclavo de la dirección que se ha configurado en bcm2835_i2c_setSlaveAddress. En el parámetro 'buf' se indica el buffer de datos a recibir, y en el parámetro 'len' la longitud de los datos a recibir (1 byte = 8 bits).
    - **uint8_t bcm2835_i2c_read (char *buf, uint32_t len**). Escribe una serie de datos en el esclavo de la dirección que se ha configurado en

bcm2835_i2c_setSlaveAddress. En el parámetro ‘buf’ se indica el buffer de datos a transmitir, y en el parámetro ‘len’ la longitud de los datos a transmitir (1 byte = 8 bits).

Entre las dos librerías presentadas se elige la librería wiringPi, debido a la facilidad de su implementación ya que no es necesario configurar los parámetros de velocidad o envío del protocolo I2C, nos es suficiente con lo que configura por defecto. El lenguaje de programación que se escoge es C, por los mismos motivos que se esgrimieron en el apartado de SPI.

El programa que se va a implementar tiene el mismo objetivo que el implementado en la Raspberry Pi para SPI. Este es el intercambio de datos básicos con la Zybo a través de la comunicación I2C.

El programa diseñado transmite al esclavo (Zybo) una serie de números de un rango del 1 al 9, es decir, 9 bytes, y leer del esclavo esos mismos bytes transmitidos. Por último, los bytes recibidos se almacenan en un buffer que se recorre para comparar lo transmitido con lo recibido. Si es igual, muestra un mensaje de éxito y sino uno de error. El flujo del programa se muestra en la siguiente figura.

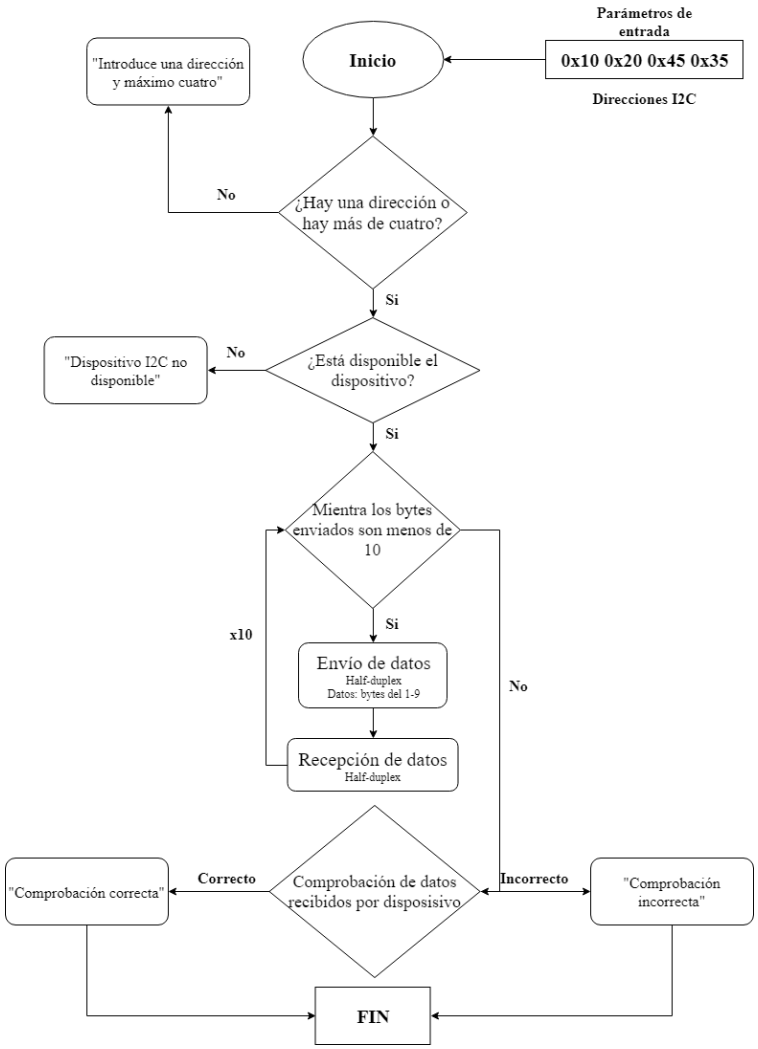


Fig. 6.22. Diagrama de flujo de la interfaz I2C

La comunicación entre maestro (Raspberry Pi) y esclavo (Zybo) utiliza el modo de funcionamiento estándar (100kbps), el modo por defecto que implementa la librería wiringPi, y los bytes se envían en transacciones individuales. En cada transacción se escribe un byte al esclavo y se lee un byte de este. Esto supone un total de 9 transacciones.

De acuerdo con los requisitos del proyecto, el sistema de control contiene un máximo de 4 conectores para los esclavos del protocolo I2C. Por ello, se necesita que el programa sea compatible con una comunicación I2C multi esclavo. Para cumplir esta directiva, la dirección de los esclavos que se introduce en el método wiringPiI2CSetup debe ser configurable a través de parámetros de entrada, así como transmitir y leer cada byte nueve veces por cada esclavo conectado.

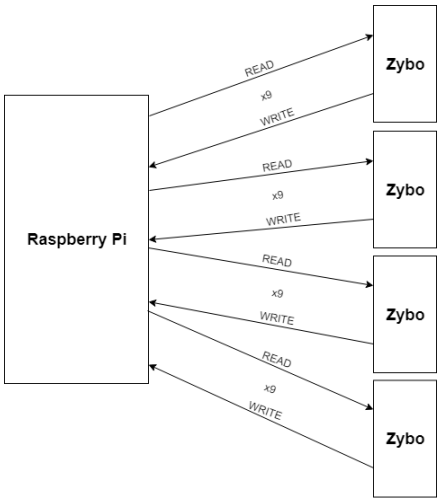


Fig. 6.23. Comunicación multi esclavo I2C

Es muy importante tener en cuenta que el protocolo I2C es half-duplex, por lo que no se pueden hacer las transacciones simultaneas con los esclavos. En consecuencia, las transacciones de cada uno se hacen de manera secuencial.

En primer lugar, para el desarrollo se necesita cargar la librería wiringPi en el programa mediante la siguiente línea:

#include wiringPI2C.h.

Posteriormente, se implementan los métodos enunciados anteriormente de acuerdo con el flujo que se define en la siguiente figura.

El programa se implementa en el archivo i2c.c a través del editor 'nano' de la interfaz de línea de comandos que ofrece PuTTY.

Finalmente, como se ha mencionado, se ha configurado un parámetro de entrada que consiste en introducir junto al comando de ejecución tantas direcciones (número entero) como esclavos estén disponibles. El comando de ejecución del programa es el siguiente:

./i2c <addr1> <addr2> <addr3> <addr4>

En conclusión, con este programa en C se puede probar una serie de intercambios de datos a través del protocolo serie I2C.

El código completo se encuentra en el Anexo O.

6.5 UART

El protocolo UART es un protocolo serie asíncrono que permite el intercambio de información que te permite trabajar en modo simplex, half-duplex y full-duplex. En este protocolo se distinguen dos líneas (más una de referencia):

- TX. Línea que se encarga en la transmisión de datos.
- RX. Línea que se encarga de recibir los datos.
- GND. Es la línea que representa masa.

UART en un principio no se puede conectar en forma de bus. La línea TX de del dispositivo se conecta con la línea RX del del otro extremo, y viceversa. Como se muestra en la siguiente figura.

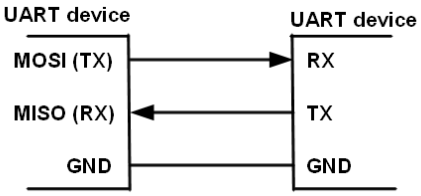


Fig. 6.24. Líneas del protocolo UART[33]

Como es un protocolo asíncrono, en la comunicación no se transmite señal de reloj. Esto provoca que ambos dispositivos deban acordar una velocidad de transmisión (baud rate) común. Este acuerdo es muy importante ya que diferente velocidad en los dispositivos a comunicar puede provocar recepciones erróneas e incluso no recibir nada.

En este sentido, antes de la comunicación también es necesario configurar el formato de trama. El formato tanto en el envío como en la recepción es como se muestra en la siguiente figura.

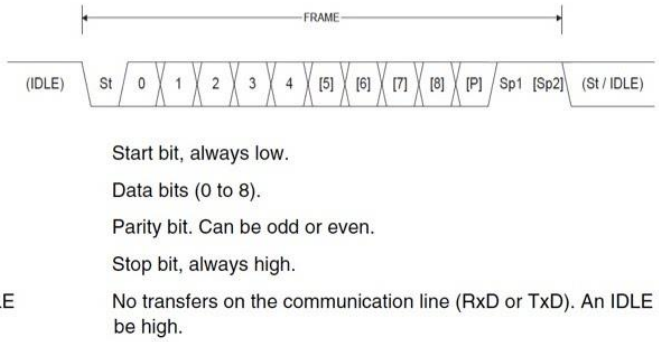


Fig. 6.25. Formato de la trama de la comunicación UART[33]

Se enumeran cada uno de los campos de la trama[34]:

- **Bit de inicio (St)**. Cuando un dispositivo UART recibe un bit de inicio, que se representa estableciendo la línea TX a nivel bajo, indica el comienzo de una transmisión de datos.
- **Datos (n)**. Seguido del bit de inicio, se envían los bits (normalmente 8) que es la información que se quiere transmitir. El mínimo de datos son 5 bits y el máximo 9 bits.
- **Paridad (P)**. Es un bit que se utiliza para detectar y corregir errores en la transmisión. Este se forma de acuerdo con el número de bits 1 que haya para transmitir. Si el número total de unos es par, el bit de paridad es 0. Por el contrario, si el número total de unos es impar, el bit de paridad es 1.

- **Bit de parada (Sp)**. Para finalizar la transmisión de los datos, el dispositivo emisor de la información establece su línea TX a nivel alto, que representa el bit de parada.
- **Reposo**. No se producen intercambios de información entre los dispositivos, está representado con las líneas a nivel alto.

Por otra parte, dentro del protocolo UART existen varios tipos de conexión según los dispositivos que se conecten:

- **Dispositivos al mismo voltaje lógico**. Cuando dos dispositivos se conectan al mismo voltaje lógico. Como se ha mencionado anteriormente, la línea TX de del dispositivo se conecta con la línea RX del del otro extremo, y viceversa. Este caso se muestra la primera conexión de la **figura**.
- **Dispositivos con distinto voltaje lógico**. Cuando dos dispositivos a conectar poseen distintos voltajes lógicos, si se conectan se puede dañar el dispositivo con menor voltaje. Para evitarlo, se implementa una etapa intermedia que reduce el voltaje de uno o aumenta el del otro. Este caso se muestra la segunda conexión de la **figura**.
- **Dispositivo con USB y dispositivo UART.** Cuando estos dos tipos de dispositivos se conectan, al igual que la conexión anterior, se necesita una etapa intermedia que se denomina convertidor TTL. Este convierte el protocolo UART a protocolo USB. Por lo tanto, se pueden utilizar dispositivos con USB que utilicen el protocolo UART. Este caso se muestra la segunda conexión de la **figura**.

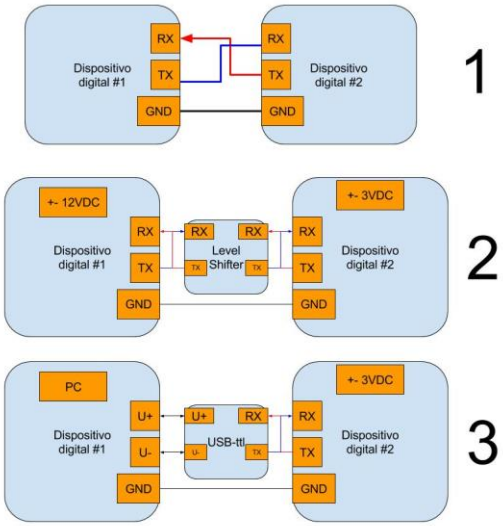


Fig. 6.26. Tipo de conexiones para dispositivos UART [34]

El protocolo UART presenta estas ventajas con respecto a otros protocolos serie[35]:

- Barato y sencillo para el intercambio de datos.
- Flexibilidad alta para la conexión y desconexión de dispositivos.

Por el contrario, también posee la siguiente serie de desventajas[35]:

- Solo pueden comunicarse entre dos dispositivos.
- La velocidad es baja y debe acordarse.

6.6 USB

Como se indica en su acrónimo, es un interfaz serial asíncrona que permite la comunicación entre varios dispositivos en modo simplex, half-duplex y full-duplex. No solo es capaz de proporcionar la comunicación entre dos dispositivos sino también su alimentación.

Existen varias versiones de la interfaz USB, pero la única diferencia desde la versión 1.0 a la 2.0 es de un aumento en la velocidad de transmisión.

Del mismo modo que existen varias versiones, también existen diversas formas de conectores USB, pero todos ellos poseen las mismas líneas en común[36]:

- +5V (VBUS).
- Datos (D-).
- Datos (D+).
- GND.

Por otra parte, la interfaz USB de puede definir en tres partes bien diferenciadas[37]:

- **Controlador host**. Es el encargado de traducir la información transmitida en el bus para que sea procesada por capas de nivel más bajo en el sistema operativo.
- **Hub raíz**. Posee los conectores físicos USB, es el encargado de detectar la conexión y desconexión de dispositivos, y, además hace de intermediario entre la comunicación de los dispositivos con el controlador.
- **Dispositivos conectados**. Son los periféricos y otro hubs que se conectan a los puertos USB.

La interfaz USB, permite el uso de la topología en estrella como se muestra en la **figura**. Esta es la forma de conexión física entre el hub, y los dispositivos conectados por puerto USB. En cuanto a la conexión lógica, solo un dispositivo puede comunicarse a la vez con el controlador host. Sin embargo, si pueden existir varios hosts permitiendo la comunicación simultanea de varios dispositivos.[37]

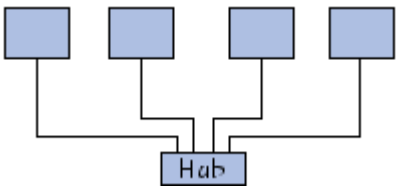


Fig. 6.27. Topología estrella[36]

En la comunicación USB se transfieren los datos en tramas. Cada dispositivo USB requiere que una porción del ancho de banda USB sea reservada durante esas tramas.[37]

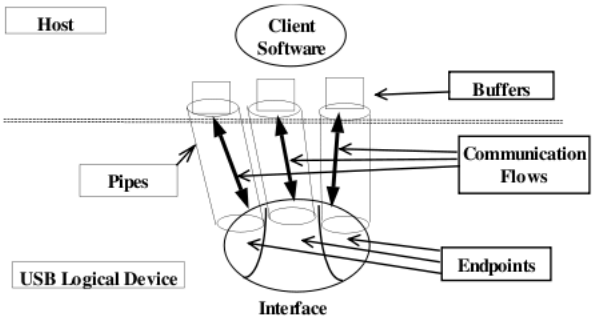


Fig. 6.28. Flujo de comunicación USB[38]

Las ventajas de la interfaz USB son las siguientes[39]:

- No es necesario alimentar el dispositivo externamente.
- Existe mecanismos para avisar al usuario de una transferencia errónea.
- Flexibilidad en la conexión y desconexión en caliente (sin reiniciar el host)
- Auto configurable.

Por el contrario, adolece de ciertas desventajas[39]:

- Solo soporta cables de un máximo de 5 metros.
- Velocidad baja con respecto a otros protocolos serie.
- Dos dispositivos no pueden comunicarse entre ellos sin pasar por el host.

6.6.1 Diseño de la comunicación mediante UART-USB

Una vez se ha explicado lo que es el protocolo USB y UART, sus especificaciones, su funcionamiento y sus fortalezas y debilidades, el siguiente paso es el diseño del programa que se encarga de utilizar el protocolo USB y el UART para el intercambio de información entre la Raspberry Pi y las Zybo.

Para ello, se divide en dos partes, una donde se explica la implementación de la interfaz USB para una comunicación serial en la Raspberry Pi, y otra donde se explica la implementación en la Zybo de la interfaz UART que se va a transmitir a través del puerto micro USB. En cada una de ellas se explica el lenguaje utilizado para el desarrollo, las librerías para la implementación de la interfaz de comunicación, el funcionamiento y los ajustes requeridos, el flujo del programa, su compilación y su ejecución.

Para la comunicación de la Zybo con la Raspberry Pi, cada una de las primeras irán conectadas por el cable USB a cada uno de los puertos de la Raspberry Pi.

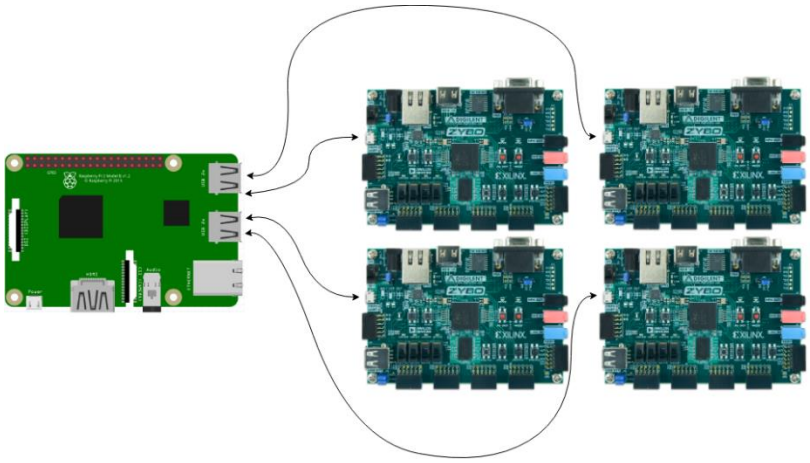


Fig. 6.29. Conexión entre Raspberry Pi y Zybo[31]

Cabe aclarar que tal y como se explica en el apartado de UART, a través de una etapa intermedia se pueden conectar la interfaz USB con la interfaz UART. El sistema operativo que implemente la interfaz USB es capaz de convertir los paquetes UART en paquetes USB a través de un software denominado “Virtual COM Port”.

6.6.1.1 USB-UART en Zybo

La Zybo implemente un puente entre el puerto micro USB y la interfaz UART (etapa intermedia). Esto permite que a través del protocolo USB se puedan intercambiar datos con la interfaz UART de la Zybo.

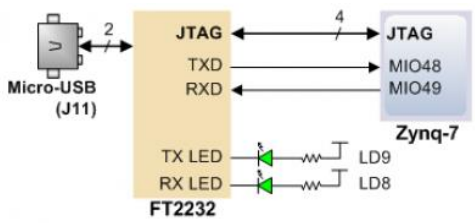


Fig. 6.30. Puente micro USB-UART[10]

Por otro lado, en la parte de la Zybo no es necesario, por nuestra parte, desarrollar nada. La interfaz UART que se requiere nos la proporciona la universidad. Esta interfaz viene compilada en un binario que se utiliza para programar la Zybo a través de la aplicación VIVADO 2015.2.

La principal función del programa es la de simular los posibles errores que puede provocar la radiación en el hardware de la Zybo mientras se introducen bytes aleatorios en su registro. Asu vez, mientras se introducen bytes en el registro la lógica del programa introduce errores aleatorios en estos. En ocasiones durante la ejecución de esta rutina, la Zybo puede quedarse congelada.

Al inicio del programa, se pide introducir un número cualquiera que el programa utilizará como semilla. Esto significa que cada número que se introduzca provoca un comportamiento diferente en la inyección de bytes e introducción de errores.

Según los bytes que se van procesando aparecen las siguientes tramas:

- **--Enter**. La trama indica que se requiere introducir una semilla para iniciar el intercambio de datos.
- **--INJ (bytes)**. La trama que empieza por esta expresión representa las secuencias de bytes que se introducen en el registro de la Zybo.
- **--INIT_T**. Indica el inicio de un archivo .vhd que genera la Zybo.
- **--FIN_T**. Indica el final de un archivo .vhd que genera la Zybo.
- **00**. Si la secuencia de bytes transcurre sin errores se envía una trama que comienza con este código.
- **22/33/44/55/66**. Si en la secuencia de bytes se produce algún error se envía una trama que comienza con este código.

Todas estas tramas se envían a través del puerto micro USB de la Zybo.

La velocidad a la que se ha configurado esta interfaz es a 115200 baudios. Las tramas de datos son de 8 bits, sin paridad (control de errores) y con un solo bit de parada.

Por otro lado, a la hora de programar la interfaz USB en la Zybo, se debe seguir el procedimiento que se incluye en el Anexo L.

6.6.1.2 USB-UART en Raspberry Pi

Como se explica en su apartado, la Raspberry Pi contiene 4 puertos USB. Cada uno de ellos con cierta limitación en la potencia que se cede al dispositivo conectado.

Basándose en el funcionamiento de la interfaz UART implementada en la Zybo, se diseña un programa en el que se implementa la interfaz serie que se comunicará a través del protocolo USB.

La librería más utilizada para el desarrollo de una interfaz serie es la librería wiringPi. Los métodos utilizados que se implementan esta librería son los siguientes[32]:

- **int serialOpen (char *device, int baud)**. Inicializa el dispositivo serie y establece la velocidad en baudios. En el parámetro 'device' se indica la dirección del dispositivo serial (/dev/ttyUSBX) y en el parámetro 'baud' se indica la velocidad que debe coincidir con la del dispositivo serie. Si la inicialización ha sido correcta devuelve un descriptor de archivo. Si el dispositivo no se ha inicializado correctamente, devuelve el valor -1.
- **int serialDataAvail (int fd)**. Devuelve el número de caracteres disponible para su lectura. En el parámetro 'fd' se introduce el descriptor de archivo devuelto por el método serialOpen. Si se produce algún tipo de error, devuelve el valor -1.
- **int serialGetchar (int fd)**. Lee un carácter del dispositivo serie. Por defecto esta llamada espera 10 segundos a la llegada de algún carácter. En el parámetro 'fd' se introduce el descriptor de archivo devuelto por el método serialOpen. Si este tiempo expira devuelve el valor -1.
- **void serialClose (int fd)**. Libera el descriptor de archivo del dispositivo. En el parámetro 'fd' se introduce el descriptor de archivo devuelto por el método serialOpen.

Los parámetros serie que se configuran por defecto para la comunicación con la Zybo son: tramas de datos con un máximo de 8 bits, sin paridad (control de errores) y con un bit de parada. En cuanto a la velocidad, como se ha comentado anteriormente debe coincidir en ambos extremos de la comunicación. En este caso, es de 11520 baudios igual en la Zybo. En este sentido, la librería wiringPi posee una función avanzada modificar estas configuraciones predeterminadas de la comunicación, dentro de ellas se han modificado dos parámetros VTIME y VMIN que afectan a lectura de los datos provenientes de la Raspberry Pi. VTIME se establece a 5, este es el tiempo que se queda la interfaz de la Raspberry Pi esperando a que llegue una nueva trama de datos. Por otro lado, VMIN se establece a 0 para que se lea siempre le máximo de bits indicado en la función de lectura (8 bits).

El funcionamiento principal del programa es la recogida de los datos que envía la Zybo a través de su puerto micro USB y su posterior almacenamiento en archivos de log. En este sentido, el programa es compatible con la recolección del máximo de 4 dispositivo de manera simultánea.

Como hemos visto en la parte de la Zybo, esta envía distintas tramas. Según se van recibiendo tramas se van escribiendo en un archivo log. Por cada dispositivo del que se recibe una trama se crea un archivo log diferente.

En consecuencia, la interfaz implementada se comporta de manera diferente en función de la trama que lea en el canal de comunicación. El procedimiento completo es: se recibe una trama, se le da un formato, se escribe en su archivo log correspondiente y se muestra en la línea de comandos. A la hora de crear el log, se crea con un nombre con este formato "**log-dd-mm-yy_hh:mm:ss-(número de dispositivo)**"

Según la trama procesada el comportamiento es el siguiente:

- **--Enter**. Esta trama indica la necesidad de la introducción de una semilla. Esta se introduce mediante una función aleatoria y se envía a la Zybo. Esta trama no se escribe en log.
- **--INJ (bytes)**. Cuando se recibe esta trama se le otorga este formato "**dd-mm-yy_hh:mm:ss INFO (trama)**" que posteriormente se almacena en el archivo log correspondiente.
- **--INIT_T**. Cuando se recibe esta trama significa que a partir de ella la Zybo va a enviar una serie de líneas que corresponden al lenguaje VHDL. Por lo tanto, se debe crear un archivo con un nombre con este formato "**VHD_Dumping_dd-mm-yy_hh:mm:ss-(número del dispositivo).vhd**". En cuanto al log, se escribe la trama con este formato "**dd-mm-yy_hh:mm:ss ERROR VHD DUMPING: ERROR (nombre del archivo .vhd)**".
- **--FIN_T**. Cuando se recibe esta trama significa que las siguientes tramas que se reciban no pertenecen al lenguaje VHDL por lo que se cierra el archivo .vhd, como se ha detectado un error apaga y enciende la Zybo a través de los pines GPIO que se han destinado para el control. Esta trama no se almacena en el log.

- **00-(bytes)**. Cuando se recibe esta trama se le otorga este formato "**dd-mm-yy_hh:mm:ss DAT 00 (trama)"** que posteriormente se almacena en el archivo log correspondiente.
- **22/33/44/55/66(bytes)**. Cuando se recibe esta trama significa que se ha producido un error en la Zybo, por lo que puede estar congelada. El programa implementado apaga y enciende la Zybo a través de los pines GPIO que se han destinado para el control. Por otro lado, a la trama se le otorga este formato "**dd-mm-yy_hh:mm:ss (código de error) (trama)**" que posteriormente se almacena en el archivo log correspondiente.

Cualquier otra trama que se reciba que no coincida con las anteriores, se almacena en el archivo log con este formato "**dd-mm-yy_hh:mm:ss INFO (trama)**".

Por otro lado, como se ha mencionado anteriormente, este programa contiene la lógica necesaria para establecer a nivel alto ('1') o nivel bajo ('0') los pines GPIO que controlan los relés y por consecuencia el paso de la corriente en los conectores donde van acopladas las Zybo.

Otra de las funciones destacadas de la interfaz es la implementación de un temporizador. Este se encarga de apagar y encender la Zybo cuando la comunicación se queda bloqueada, es decir, cunado la Raspberry Pi no recibe nada o ha fallado la ejecución del programa o porque la Zybo no trasmite nada.

La última característica implementada, es la configuración por parámetros de entrada, es decir, parámetros que se introducen en el comando que ejecuta el programa. Estos parámetros son la dirección USB de la Zybo(/dev/ttyUSBX) y el pin GPIO asociado al conector donde se alimenta la Zybo que se determina con un número del 1 al 4, siendo 1 el conector de arriba a izquierda y 4 el de abajo a la izquierda.

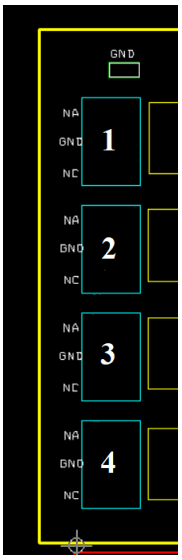


Fig. 6.31. Conectores numerados para las Zybo del Sistema de Control

En relación con el párrafo anterior, para ejecutar el programa antes es necesario compilarlo con el compilador GCC. Junto con la inclusión de la parte serie de la librería wiringPI mediante la línea:

#include <wiringSerial.h>

Las líneas de código en C están contenidas en los archivos serialC.c (donde se encuentra la parte principal del programa), y functions.c/functions.h (donde se implementa las funciones de la interfaz).

Finalmente, como se ha mencionado, se han configurado dos parámetros de entrada que se define junto con el binario el primer parámetro debe ser la dirección USB de la Zybo con este formado /dev/ttyUSBX y el siguiente un número del 1 al 4 . El comando de ejecución del programa es el siguiente:

./serialC /dev/ttyUSBX <pin de control>

Para ejecutar los 4 dispositivos a la vez, no es posible hacerlo con una sola ejecución del programa, sino que se tienen que abrir una ventana de comandos distinta por cada Zybo conectada.

En conclusión, esta parte completaría la comunicación serie USB-UART entre el sistema de control y las Zybo.

El código completo se encuentra en el Anexo Q.

Comunicación serial UART en Rasberry Pi

Uno de los protocolos serie implementado en la Raspberry Pi es el protocolo UART. La Raspberry Pi posee implementado un módulo UART al que están conectados dos tipos diferentes de puertos:

- **Pines GPIO serial**. Estos pines son la línea RX y TX típica de una comunicación UART. Solo están implementados dos pines, por lo que solo se puede conectar un dispositivo. Esto dos pines GPIO son el 8 (TX) y el 10 (RX)
- **Puertos USB**. La Raspberry Pi posee cuatro puertos USB, compatibles para la conexión de un dispositivo UART mediante un convertidor USB-TTL. En este caso, como se mencionará posteriormente la Zybo se conecta a otro puerto micro-USB por lo que la comunicación UART llevará a cabo a través de un protocolo USB.

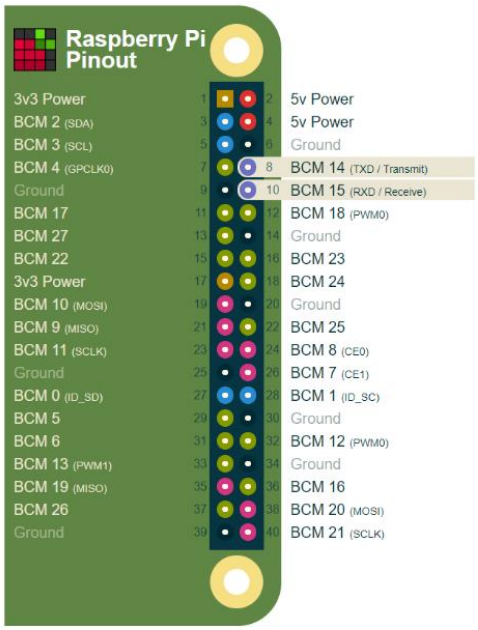


Fig. 6.32. Pines reservados para UART[40]

El programa implementado para el protocolo UART es el mismo que el del protocolo USB. Ya que ambos protocolos pueden conectarse entre sí.

En este sentido, tanto la lógica del programa como lo métodos son totalmente compatibles con UART. La única diferencia es que la interfaz UART se encuentra en '/dev/ttyAMA0'.

6.7 Pruebas

Finalizado el diseño de las interfaces, en esta última fase se llevan a cabo una serie de pruebas que evidencian el funcionamiento de cada uno de estos protocolos. Estas se dividen según el protocolo que se vaya a ejecutar.

6.7.1 Pruebas SPI

El propósito de esta prueba es comprobar el funcionamiento de una comunicación a través del protocolo SPI entre la Zybo y el sistema de control. Cabe destacar que la interfaz que se ha implementado de este protocolo solo soporta un máximo de dos dispositivos.

Los elementos utilizados en esta prueba son dos Zybo dispuestas una encima de otra, el sistema de control y un ordenador. Cabe destacar que al estar ante una arquitectura maestro-esclavo, como se ha mencionado anteriormente, el sistema de control es el maestro y la Zybo el esclavo.

El montaje que se lleva a cabo consta de dos pares de cables que se utilizan como conexión de la alimentación de las Zybo (configuradas en modo alimentación por jack) a las clemas del sistema de control (primer y segunda). Y a continuación para cada una de ellas se conectan cinco cables desde su puerto pmod JE al conector de cinco pines del sistema de control (MISO, MOSI, SCLK, SSX y GND) para los canales de la comunicación SPI. Por otra parte, se conecta la Raspberry Pi a la placa a través de un cable de 40 pines y al ordenador a través de un cable Ethernet.

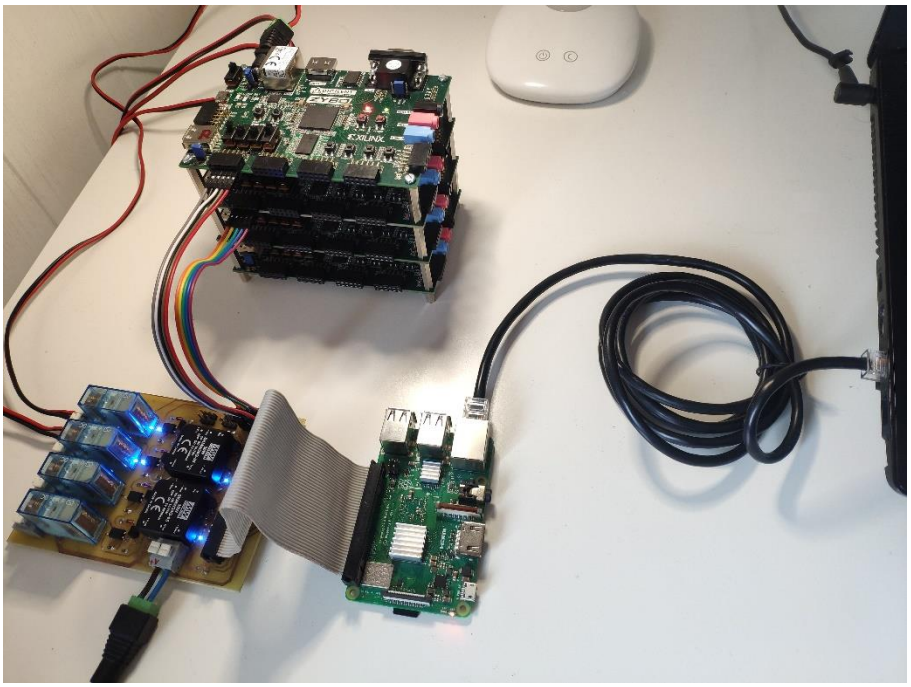


Fig. 6.33. Montaje para pruebas SPI

Antes de realizar las pruebas se llevan a cabo una serie de configuraciones previas. Las Zybo se programan en JTAG con la interfaz SPI mediante el procedimiento que se explica en el Anexo L.

Por otro lado, el programa es capaz gestionar un dispositivo por ejecución por lo que se abren dos sesiones con PuTTY. En una de ellas se procede a compilar el programa desarrollado para la Raspberry mediante el siguiente comando:

gcc -g -w -Wall -O3 spi.c -o spi -lbcm2835

Finalizada la configuración, se procede a realizar la prueba de la comunicación SPI. Para ello se ejecuta el programa para la Raspberry Pi mediante el comando:

sudo ./spi -c <chip selector> -p <pin de control>

En la primera sesión, en el parámetro de entrada '-c' se introduce un 0 que es el canal SS SPI de la primera Zybo y en el parámetro '-p' un 1 que es el pin correspondiente a la posición en la que está conecta la Zybo en el sistema de control, la 1ª. En cambio, en la segunda sesión el parámetro de entrada '-c' se introduce el número 1 que es el canal SS SPI de la segunda Zybo y en el parámetro '-p' un 2.

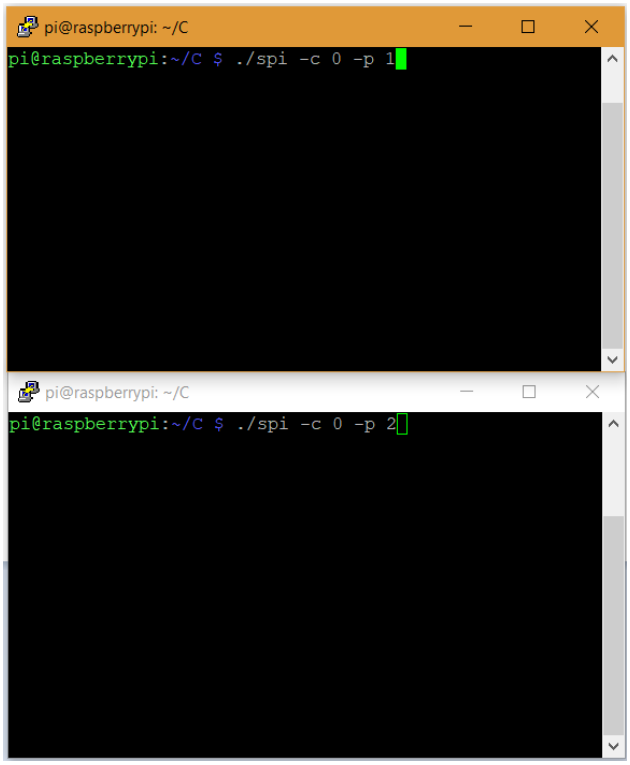


Fig. 6.34. Sesión PuTTY para comunicación SPI

Se pulsar ‘Enter’, y se comienza la transferencia de bytes entre la Zybo y el sistema de control. El resultado obtenido se muestra en la siguiente figura.

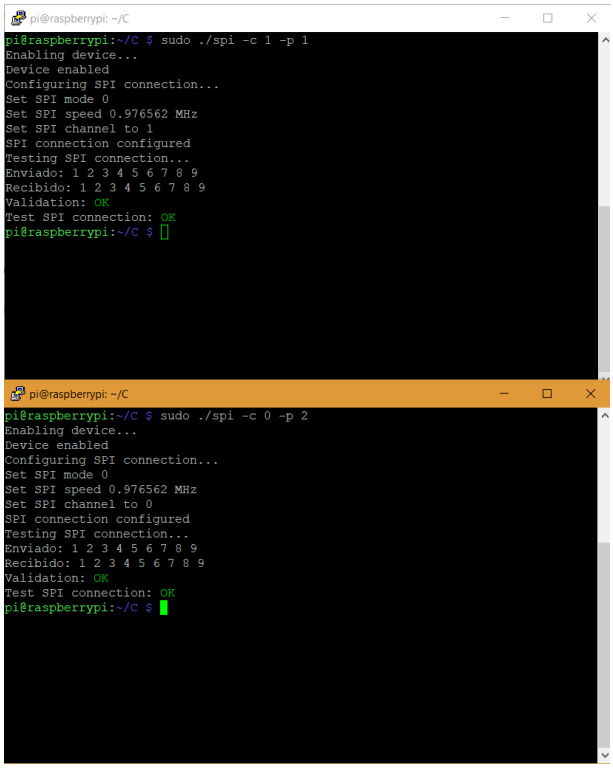


Fig. 6.35. Resultado correcto en la transferencia SPI

Se observa que la lógica de comprobación que incluye el programa ha lanzado un resultado satisfactorio, es decir, lo enviado a la Zybo es igual que lo recibido de ella.

En cambio, si se produjera algún tipo de error en la transferencia se muestra el siguiente resultado.

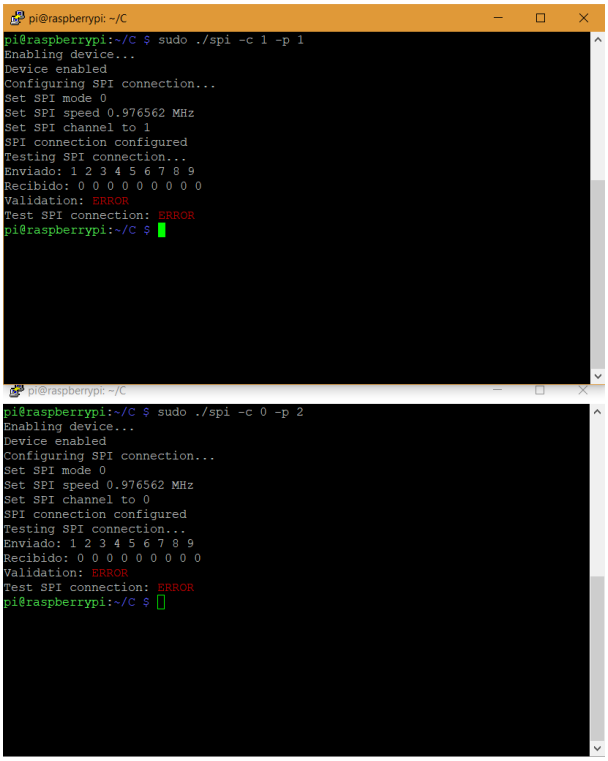


Fig. 6.36. Resultado erróneo en la transferencia SPI

6.7.2 Pruebas I2C

El objetivo de esta prueba es comprobar el funcionamiento de una comunicación a través del protocolo I2C entre la Zybo y el sistema de control. A diferencia de la anterior, la interfaz que se ha implementado de este protocolo solo soporta un máximo de cuatro dispositivos.

Los elementos utilizados en esta prueba son un clúster de cuatro Zybo dispuestas una encima de otra, el sistema de control y un ordenador. Cabe destacar que al estar ante una arquitectura maestro-esclavo, como se ha mencionado anteriormente, el sistema de control es el maestro y la Zybo el esclavo.

El montaje que realizado consta de cuatro pares de cables que se utilizan como conexión de la alimentación de las Zybo (configuradas en modo alimentación por jack) a las clemas del sistema de control. Y a continuación para cada una de ellas se conectan tres cables desde su puerto pmod JE al conector de tres pines del sistema de control (SCL, SDA y GND) para los canales de la comunicación I2C. Por otra parte, se conecta la Raspberry Pi a la placa a través de un cable de 40 pines y al ordenador a través de un cable Ethernet.

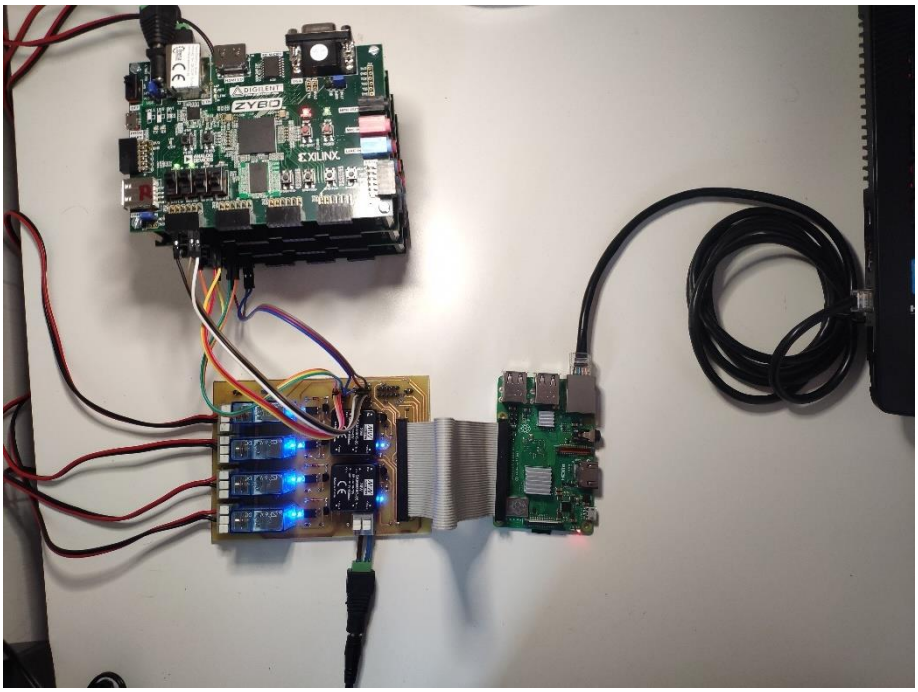


Fig. 6.37. Montaje para pruebas SPI

Antes de realizar las pruebas se llevan a cabo una serie de configuraciones previas. Las Zybo se programan en JTAG con la interfaz I2C mediante el procedimiento que se explica en el Anexo L.

Por otro lado, el programa es capaz gestionar un máximo de cuatro dispositivos por ejecución por lo que se abre una sola sesión con PuTTY. A continuación, se procede a compilar el programa desarrollado para la Raspberry mediante el siguiente comando:

gcc -g -w -Wall -O3 i2c.c -o i2c -lwiringpi

Después, para comprobar en que direcciones I2C están disponibles las Zybo se introduce el siguiente comando:

i2cdetect -y -1

El resultado del comando se muestra en la siguiente figura.

```
pi@raspberrypi:~/C $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: 60 -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: 70 -- -- -- -- -- -- --
```

Fig. 6.38. Direcciones I2C de las Zybo

Finalizada la configuración, se procede a realizar la prueba de la comunicación I2C. Para ello se ejecuta el programa para la Raspberry Pi mediante el comando:

./i2c 0x48 0x60 0x68 0x70

Cómo se ha explicado en su sección correspondiente, en la ejecución del programa se introducen como parámetros de entrada, cada una de las direcciones I2C que configurada en las Zybo mediante sus 'switches' que se muestra en la figura anterior.

Al ejecutar el comando comienza la transferencia de bytes entre la Zybo y el sistema de control. El resultado obtenido se muestra en la siguiente figura.

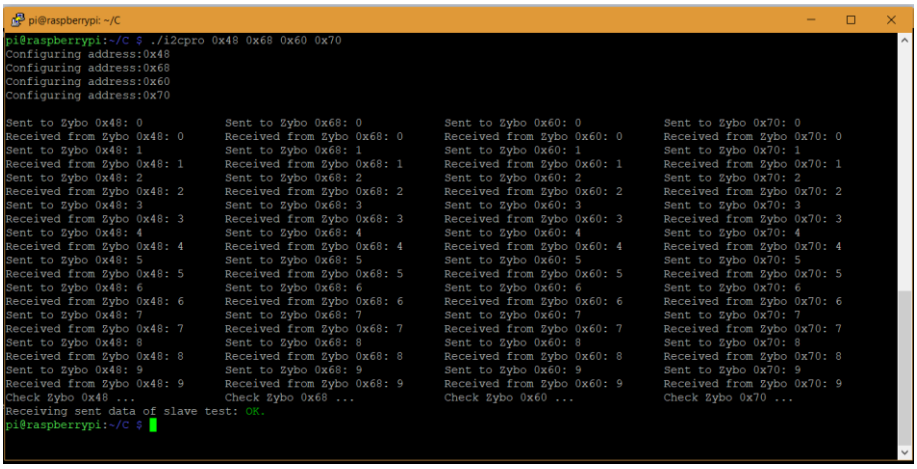


Fig. 6.39. Resultado correcto en la transferencia I2C

Se observa que la lógica de comprobación que incluye el programa ha lanzado un resultado satisfactorio, es decir, lo enviado a la Zybo es igual que lo recibido de ella.

En cambio, si se produjera algún tipo de error en la transferencia este sería el resultado lanzado por el programa.

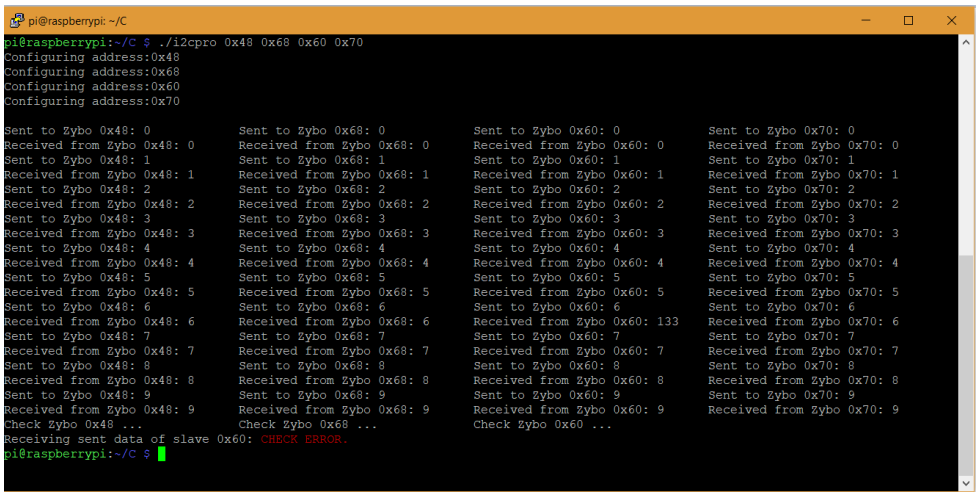


Fig. 6.40. Resultado erróneo en la transferencia I2C

6.7.3 Pruebas USB-UART

El objetivo principal de esta prueba es comprobar el funcionamiento de una comunicación USB-UART entre la Zybo y el sistema de control. La interfaz que se ha implementado soporta un máximo de cuatro dispositivos.

Los dispositivos utilizados en esta prueba son un clúster de cuatro Zybo dispuestas una encima de otra, el sistema de control y un ordenador.

El montaje realizado consta de cuatro pares de cables que se utilizan como conexión de la alimentación de las Zybo (configuradas en modo alimentación por jack) a las clemas del sistema de control. A esto se le añade la conexión de cuatro cables USB desde su puerto micro USB a los puertos USB que implementa la Raspberry Pi. Por otra parte, en el sistema de control se conecta la Raspberry Pi a la placa a través de un cable de 40 pines y al ordenador a través de un cable Ethernet.

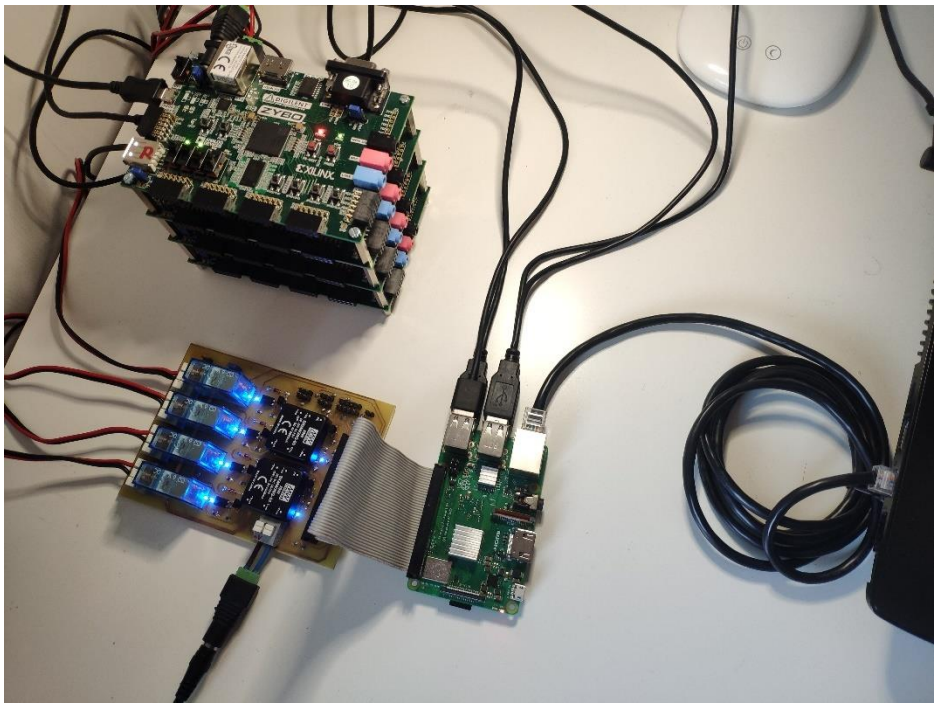


Fig. 6.41. Montaje prueba USB-UART

Antes de realizar las pruebas se llevan a cabo una serie de configuraciones. Las Zybo se programan en QSPI con la interfaz USB-UART mediante el procedimiento que se explica en el Anexo L. Por otro lado, el programa es capaz gestionar un dispositivo por ejecución por lo que se abren cuatro sesiones con PuTTY. En una de ellas se procede a compilar el programa desarrollado para la Raspberry mediante el siguiente comando:

gcc -g -w -Wall -O3 serialC.c funtion.c util.c -o serialC -lm -lwiringPi –debug

Finalizada la configuración, se procede a realizar la prueba de la comunicación USB-UART. Para ello se ejecuta el programa para la Raspberry Pi mediante el comando:

./serialC /dev/ttyUSBX <pin de control>

Junto al comando de ejecución del binario se introduce la dirección del dispositivo USB que asigna la Zybo y el pin GPIO correspondiente al control de la alimentación de esta. Debido a que en función del orden en el que se conectan dispositivos USB se asigna una dirección diferente. Se configura un archivo que contiene una serie de reglas para que a cada dispositivo se le asigne siempre la misma dirección USB.

En la ejecución comienza la transferencia de tramas entre la Zybo y el sistema de control. El resultado obtenido se muestra en la siguiente figura.

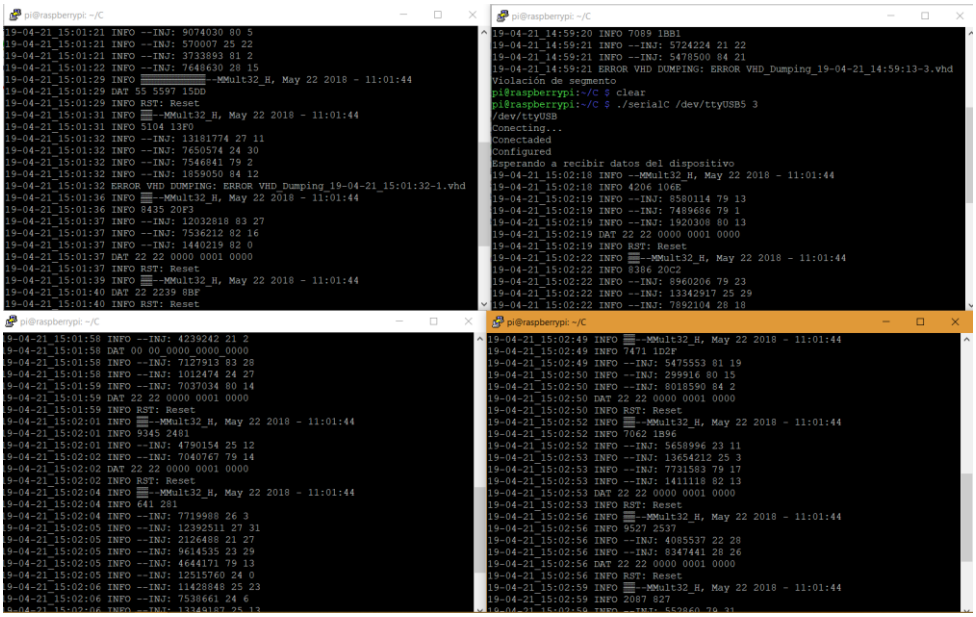


Fig. 6.42. Resultado correcto en la transferencia USB-UART

Se observa que se muestra por pantalla todas las tramas que se reciben de las Zybo y se guardan en su log correspondiente. Además, cada vez que se recibe un error de la Zybo, el sistema de control la apaga y la enciende.

En conclusión, de acuerdo con los resultados obtenidos el comportamiento de la comunicación coincide con lo establecido en los requisitos.

7 PRESUPUESTO

En esta sección se elabora una tabla con el presupuesto del proyecto en el que se valor el gasto en hardware, en software y la mano de obre relacionada con el diseño, fabricación y montaje.

TABLA 7.1. PRESUPUESTO DEL PROYECTO

	Cantidad	Modelo	Precio (€)
Hardware			
Fuente de alimentación	1	TP Link 48V 1.25A 60W	20
Switch	1	PS504	40
Raspberry Pi	1	3B+/B+	35
Zybo	4	Zynq-7000	400
Relé	4	Finder 40.51 6V	18
Optoacoplador	4	TPC817C	2.4
Resistencia	12	470/1k	0.2
Led	6	APHHS1005LQBC	3.5
Transistor	6	BC547B	0.9
Diodo	4	1N4148	0.24
Convertidores	2	SKMW30G-05	66
Clema 3 puertos	4	WAGO 235	2
Clema 2 puertos	1	WAGO 235	0.4
Regleta de 3 pines	4	Macho 2.54mm	0.20
Regleta de 2 pines	3	Macho 2.54mm	0.15
Regleta de 5 pines	2	Macho 2.54mm	0.14
Regleta de 2x20 pines	1	Macho 2.54mm	0.5
Cable de 40 pines	1	0.2m	9
Conector Jack	5	Macho	2
Cables USB	4	2m	20
Jumper	1	-	0.5
Cables de alimentación para Zybo	4	2m	5
Cables para comunicación	24	2m	20
PCB	1	-	30
Subtotal	-	-	667
Software			
Multisim	1 licencia	13	400
Vivado	1 licencia	2015.2	2665
OrCAD	1 licencia	10.3	1000
Subtotal	-	-	4065

Mano de obra			
Diseño del circuito	-	-	25
Diseño de la PCB	-	-	20
Diseño del software	-	-	30
Fabricación de la PCB	-	-	50
Montaje	-	-	10
Subtotal	-	-	135
Total			4867

8 CONCLUSIÓN Y FUTURAS MEJORAS

En este último capítulo se señala cuáles han sido las dificultades que se han presentado a lo largo del proyecto. Así como, que objetivos propuestos al principio del documento se han conseguido y de qué forma. Por último, se cierra el capítulo, con una serie de propuestas para futuras mejoras del proyecto.

8.1 Dificultades y problemas

Durante la elaboración del proyecto se han presentado una serie de dificultades que es necesario mencionar, debido a que han sido factores influyentes a largo de la realización de este. Se han enumerado las más importantes por capítulo:

- En cuanto al diseño del sistema de control, en concreto el circuito de la placa la parte más complicada fue investigar una forma funcional de implementar el conjunto relé-optoacoplador.
- En cuanto a la fabricación de la placa, quiero destacar la dificultad que tuve a la hora del trazado de las pistas. Hice aproximadamente 13 diseños de enrutado. Y la otra dificultad también muy importante, fue el soldado de componentes SMD en la placa. Esto se debe a que era la primera vez que los soldaba, especialmente los leds que son muy pequeños.
- En cuanto al diseño de software, las dificultades principales que encontré fue la implementación de las interfaces SPI e I2C en la Zybo sobre todo en esta última por la cantidad de estados que tiene.

8.2 Conclusiones

Como punto final, el diseño que se ha propuesto perseguía dos objetivos reducir a lo indispensable el cableado del montaje del experimento y mejorar la eficiencia del experimento mediante la centralización y automatización del sistema de control.

Con el propósito de mejorar la eficiencia del experimento, se ha centralizado el control y se ha otorgado la suficiente autonomía al sistema de control para una vez comenzado el experimento no tener la necesidad de disponer de control humano. Además, a través de la conexión Ethernet entre el ordenador se dispone un control total de las alimentaciones del sistema de control y los dispositivos de manera manual. Por lo que se ha alcanzado el primer objetivo con creces.

Otro aspecto importante, es la reducción del cableado a lo indispensable. Solo se utiliza un transformador, en vez de 5, para alimentar todo el experimento, tanto el sistema de control como las Zybo. En cambio, se siguen necesitando de 4 pares de cables por Zybo, de cuatro cables USB si se quiere utilizar el protocolo USB-UART, de 12 cables si se quiere utilizar el protocolo I2C (8 sin masas), de 10 cables si se quiere utilizar el protocolo SPI (8 sin masas) y de 2 cables si se quiere utilizar el protocolo UART.

Por lo tanto, parece que la inclusión de nuevos protocolos haya producido que el cableado de comunicación se vuelva más tedioso. Sin embargo, si estos protocolos hubieran estado disponibles en anteriores diseños, el cableado hubiera sido mucho más complicado. Por lo que la cumplimentación de este segundo objetivo ha sido completa.

En definitiva, de los dos objetivos marcados se han llegado a conseguir ambos de la manera esperada.

8.3 Futuras mejoras

Finalizando el documento, se sugieren ciertas propuestas para posteriores versiones del sistema de control que completarían y mejorarían aún más su función.

En la parte hardware, en cuanto al diseño del circuito se podría añadir un medidor de corriente que a partir de cierto umbral se interrumpiera la alimentación. El objetivo es proporcionar mayor seguridad al sistema de control y a las Zybo ante picos de corriente. Otra mejora sería utilizar un cable de 40 pines que a su vez te permita conectar cables individuales a los GPIO de la Raspberry Pi a través del conector de la placa, por si se necesitaran pines GPIO auxiliares. Una última mejora de esta sección sería eliminar el bloque de alimentación y conversión y alimentar la Raspberry Pi directamente a través un Ethernet POE (Cable Ethernet que transporta la comunicación y la alimentación) proporcionada a partir de un switch. Y a su vez, a través de los pines de 5V de la Raspberry Pi alimentar el circuito. Sin embargo, el problema es que la Raspberry Pi 3B+ es el único modelo compatible con este tipo de tecnología.

En cuanto a la fabricación, una de las mejoras estaría enfocada a reducir el espacio que ocupan los componentes de agujero pasante utilizando su alternativa (si la tiene) SMD, aumentando el espacio de trazado. Por ejemplo, los transistores se podrían utilizar SMD, los diodos e incluso los optoacopladores. Otra de las mejoras sería implementar ciertos componentes SMD en la parte inferior de la placa y así tener mayor espacio libre para componentes en la parte de arriba. Otra mejora importante sería la impresión de una capa de masa común para ahorrarnos el trazado de las pistas de masa y aumentar el espacio de trazado.

En lo que respecta al diseño software, las mejora que recomendaría implementar son en la comunicación I2C y SPI. Esta consiste en utilizar para el I2C y SPI para el intercambio de otro tipo de datos, que no se transmiten a través del protocolo USB-UAR,T entre la Zybo y el sistema de control. Una última mejora sería implementar un programa en la Raspberry Pi con la interfaz capaz de gestionar de manera simultánea las tres comunicaciones. En este programa se podría activar o desactivar protocolos, visualizar el intercambio de datos en cada Zybo y protocolo, disponer control manual de la alimentación de la Zybo y el sistema de control, y de los pines GPIO, visualizar el estado de las Zybo (activas, tensión, corriente, potencia, temperatura), entre otras. En cambio, en la parte de la Zybo se propone aglutinar las tres interfaces en un solo programa.

En conclusión, a pesar de que el diseño propuesto soluciona varios problemas de montajes anteriores. Aún queda margen para implementar futuras mejoras, con el objetivo de recolectar un mayor número de datos, mayor versatilidad por el canal en el que se reciben los datos o que tipos en cada uno, flexibilidad y sencillez en el montaje, autónomo y con capacidad de autorrecuperarse ante errores.

BIBLIOGRAFIA

[1] «FAQs - Raspberry Pi Documentation», *Raspberry Pi*, 2014. [En línea]. Disponible en: https://www.raspberrypi.org/documentation/faqs/#power. [Accedido: 02-jun-2019]

[2] «Raspberry Pi tabla técnica completa con todos los modelos.», *Raspberry para torpes*, 2018. [En línea]. Disponible en: https://raspberryparatorpes.net/raspberry-pi-tabla-tecnica-completa/. [Accedido: 05-may-2019]

[3] A. Aqeel, «Introduction to Raspberry Pi 3 B+», *The Engineering Projects*, 2018. [En línea]. Disponible en: https://www.theengineeringprojects.com/2018/07/introduction-to-raspberry-pi-3-b-plus.html. [Accedido: 05-may-2019]

[4] «New Raspberry Pi Model B+ with Better Audio | audioXpress», *Audio Xpress*, 2014. [En línea]. Disponible en: https://www.audioxpress.com/news/New-Raspberry-Pi-Model-B-with-better-audio. [Accedido: 16-jun-2019]

[5] «Burn SD cards with Etcher - The MagPi MagazineThe MagPi Magazine», *Raspberry Pi*, 2017. [En línea]. Disponible en: https://www.raspberrypi.org/magpi/pi-sd-etcher/. [Accedido: 16-jun-2019]

[6] «GPIO - Raspberry Pi Documentation», *Raspberry Pi*, 2019. [En línea]. Disponible en: https://www.raspberrypi.org/documentation/usage/gpio/. [Accedido: 05-may-2019]

[7] «I2C at Raspberry Pi GPIO Pinout», *Pinout*, 2018. [En línea]. Disponible en: https://pinout.xyz/pinout/i2c. [Accedido: 29-may-2019]

[8] «Raspberry Pi GPIO Pinout», *Pinout*, 2019. [En línea]. Disponible en: https://pinout.xyz/. [Accedido: 05-may-2019]

[9] «Zybo [Reference.Digilentinc]», *Digilent Inc*, 2019. [En línea]. Disponible en: https://reference.digilentinc.com/reference/programmable-logic/zybo/start. [Accedido: 09-jun-2019]

[10] «Zybo Reference Manual [Reference.Digilentinc]», *Digilent*, 2014. [En línea]. Disponible en: https://reference.digilentinc.com/reference/programmable-logic/zybo/reference-manual. [Accedido: 02-jun-2019]

[11] «Comunicación serie», *IBM*, 2019. [En línea]. Disponible en: https://www.ibm.com/support/knowledgecenter/es/ssw_aix_72/com.ibm.aix.networkcomm/asynch_serialcomm.htm. [Accedido: 16-jun-2019]

[12] «Transmisión de datos en Serie y Paralelo – PC-Solución», *PC-Soucion*, 2018. [En línea]. Disponible en: https://pc-solucion.es/2018/04/03/transmision-de-datos-en-serie-y-paralelo/. [Accedido: 16-jun-2019]

[13] «¿Cómo funciona el protocolo SPI? | Panama Hitek», *PanamaHitek*, 2014. [En línea]. Disponible en: http://panamahitek.com/como-funciona-el-protocolo-spi/. [Accedido: 31-may-2019]

[14] «Serial Peripheral Interface - Wikipedia, la enciclopedia libre», *Wikipedia*, 2018. [En línea]. Disponible en: https://es.wikipedia.org/wiki/Serial_Peripheral_Interface#Desventajas. [Accedido: 31-may-2019]

[15] «Pic Pic18f4550 Spi | Pic», *Electronic wings*, 2018. [En línea]. Disponible en: https://www.electronicwings.com/pic/pic18f4550-spi. [Accedido: 16-jun-2019]

[16] «Protocolo de comunicación SPI – Vida Embebida», *Vida Embebida*, 2017. [En línea]. Disponible en: https://vidaembebida.wordpress.com/2017/02/08/protocolo-de-comunicacion-spi/. [Accedido: 31-may-2019]

[17] «ADALM2000 - Debugging SPI - Blogs - Virtual Classroom for ADI University Program - EngineerZone», *EngineerZone*, 2019. [En línea]. Disponible en: https://ez.analog.com/university-program/b/blogs/posts/adalm2000-spi-debug. [Accedido: 31-may-2019]

[18] «El bus SPI en Arduino», *Luis Llamas*, 2016. [En línea]. Disponible en: https://www.luisllamas.es/arduino-spi/. [Accedido: 31-may-2019]

[19] «SPI at Raspberry Pi GPIO Pinout», *Pinout*, 2018. [En línea]. Disponible en: https://pinout.xyz/pinout/spi. [Accedido: 30-may-2019]

[20] «SPI - Raspberry Pi Documentation», *Raspberry Pi*, 2019. [En línea]. Disponible en: https://www.raspberrypi.org/documentation/hardware/raspberrypi/spi/README.md. [Accedido: 05-may-2019]

[21] «SPI Library | Wiring Pi», *Wiring Pi*, 2013. [En línea]. Disponible en: http://wiringpi.com/reference/spi-library/. [Accedido: 05-may-2019]

[22] «bcm2835: SPI access», *AirSpayce*, 2012. [En línea]. Disponible en: http://www.airspayce.com/mikem/bcm2835/group__spi.html#gad58ade25f8eaaacd4d9249f51af4d1f3. [Accedido: 05-may-2019]

[23] «fpga - UART Receiver Sampling Rate», *Stack Exchange*, 2015. [En línea]. Disponible en: https://electronics.stackexchange.com/questions/207870/uart-receiver-sampling-rate. [Accedido: 05-may-2019]

[24] «Convoluted Convolution – Sanjay Nair – Medium», *Medium*, 2018. [En línea]. Disponible en: https://medium.com/@nirespire/convoluted-convolution-829e953813cf. [Accedido: 17-jun-2019]

[25] «¿Que es I2C?», *Estuelectronic*, 2012. [En línea]. Disponible en:

https://estuelectronic.wordpress.com/2012/11/06/que-es-i2c/. [Accedido: 16-jun-2019]

[26] L. Llamas, «El bus I2C en Arduino», *Luis Llamas*, 2016. [En línea]. Disponible en: https://www.luisllamas.es/arduino-i2c/. [Accedido: 16-jun-2019]

[27] «Descripción y funcionamiento del Bus I2C | Robots Didácticos», *Robots Didácticos*, 2018. [En línea]. Disponible en: http://robots-argentina.com.ar/didactica/descripcion-y-funcionamiento-del-bus-i2c/. [Accedido: 05-may-2019]

[28] «Basics of the I2C Communication Protocol», *Circuit Basics*, 2016. [En línea]. Disponible en: http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/. [Accedido: 28-may-2019]

[29] M. Morales, «Protocolo I2C - Fundamentos de aprendizaje. ¿Cómo funciona? - Clases TBem.», *TeslaBem*, 2017. [En línea]. Disponible en: https://teslabem.com/nivel-intermedio/fundamentos-del-protocolo-i2c-aprende/. [Accedido: 16-jun-2019]

[30] «I2C Bus Specification», *I2C Info*, 2013. [En línea]. Disponible en: https://i2c.info/i2c-bus-specification. [Accedido: 16-jun-2019]

[31] «I2C Library | Wiring Pi», *Wiring Pi*, 2012. [En línea]. Disponible en: http://wiringpi.com/reference/i2c-library/. [Accedido: 29-may-2019]

[32] «bcm2835: I2C access», *AirSpayce*, 2012. [En línea]. Disponible en: https://www.airspayce.com/mikem/bcm2835/group__i2c.html#ga6b0495cb138bd5eb26965f5a00b08dc0. [Accedido: 30-may-2019]

[33] «Puertos y Buses 1: I2C y UART - Geeky Theory», *Geeky Theory*, 2015. [En línea]. Disponible en: https://geekytheory.com/puertos-y-buses-1-i2c-y-uart. [Accedido: 01-jun-2019]

[34] «Puerto Serial - protocolo y su teoría - HETPRO/TUTORIALES», *Hetpro*, 2017. [En línea]. Disponible en: https://hetpro-store.com/TUTORIALES/puerto-serial/. [Accedido: 01-jun-2019]

[35] «UART vs SPI vs I2C Diferencias entre protocolos. - Drouiz», *Drouiz*, 2018. [En línea]. Disponible en: https://www.drouiz.com/blog/2018/06/25/uart-vs-spi-vs-i2c-diferencias-entre-protocolos/. [Accedido: 01-jun-2019]

[36] «USB (Bus de serie universal)», *CCM*, 2017. [En línea]. Disponible en: https://es.ccm.net/contents/407-usb-bus-de-serie-universal. [Accedido: 01-jun-2019]

[37] «Capítulo 3 Introducción a la arquitectura y protocolo del Bus Serie Universal 3.1. Introducción», 2019 [En línea]. Disponible en: http://bibing.us.es/proyectos/abreproy/11419/fichero/Volumen+1%252F03+Tem

a+3.pdf. [Accedido: 01-jun-2019]

[38] «Universal Serial Bus - OSDev Wiki», *OsDev*, 2017. [En línea]. Disponible en: https://wiki.osdev.org/Universal_Serial_Bus. [Accedido: 01-jun-2019]

[39] «Advantages of USB | disadvantages of USB interface», *RF Wireless World*, 2012. [En línea]. Disponible en: http://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-USB.html. [Accedido: 01-jun-2019]

[40] «Multisim Download - National Instruments», *National Instruments*, 2017. [En línea]. Disponible en: http://www.ni.com/es-es/support/downloads/software-products/download.multisim.html#306441. [Accedido: 09-jun-2019]

[41] I. Pérez, «Introducción a la simulación de circuitos electrónicos», *OCW UC3M*, p. 12, 2015 [En línea]. Disponible en: http://ocw.uc3m.es/tecnologia-electronica/componentes-y-circuitos-electronicos/practicas-1/OCW-CCE_P1_Introduccion_simulacion_circuitos_electronicos.pdf. [Accedido: 09-jun-2019]

[42] «Start Your First Schematic Design in OrCAD Capture», *OrCAD*, 2019. [En línea]. Disponible en: https://www2.orcad.com/Extranet/96712/forms.aspx?msgid=f61b0efc-1eda-40b4-8872-9f2351fa3777&LinkID=CH00096712eR00000065AD. [Accedido: 09-jun-2019]

[43] «7: Final PCB Design Snapshot using Orcad Layout plus. | Download Scientific Diagram», *Research Gate*, 2014. [En línea]. Disponible en: https://www.researchgate.net/figure/Final-PCB-Design-Snapshot-using-Orcad-Layout-plus_fig13_269874187. [Accedido: 09-jun-2019]

ANEXO A. RESUMEN EN INGLÉS

This document presents the design of a control system for devices used in radiation experiments. The objective is to improve the design of the assemblies that have been used in previous radiation experiments with these devices.

The experiment consists of stacking a series of FPGA devices called Zybo in a laboratory prepared to work with radiation. In front of the Zybo there is a machine that radiates beams of neutrons and protons pointing towards its SoC.

The neutrons, having a big mass, go through each of the devices producing holes in the tracks. This can cause logical changes where bits go from "1" to "0", and vice versa.

Protons, on the other hand, have much less mass and are not able to pass through devices. In these cases only one Zybo is placed to radiate. Like neutrons, they can produce holes in the tracks. This can cause changes at a logical level where bits go from "1" to "0", and vice versa.

The Zybo are connected to a control system. This consists of a relay module and a Raspberry Pi. In the relay module is connected to the power cables of the Zybo and each of its transformers. In this module are connected several pins GPIO Raspberry Pi to control the state of the relays (switches), ie whether or not provide power to the Zybo.

With regard to the Raspberry Pi, apart from controlling the relays it also provides the interface to communicate via USB with the UART interface of the Zybo. Each of these is connected to the four USB ports contained in the Raspberry Pi. The Raspberry Pi collects data from the Zybo and restarts it through the relays if it detects an error.

This assembly presents several problems. The first problem is the amount of power supplies used by each Zybo and another for the Raspberry Pi, and the other is the amount of cables used, two power cables for each Zybo 4 cables from the GPIO pins and two power cables from the relay module. This is multiplied by two if you want to use two or more control systems.

Therefore the main objective of this document is to present an alternative that addresses the deficiencies above. This means that in order to improve the assembly it is necessary to improve the design of the control system that is part of the experiment.

The control system used so far consists of a relay module (separate or hat), and the brain of the control system which is a Raspberry Pi. Therefore, the best that are introduced are the design of a board with the necessary relays, power and connectors along with a Raspberry Pi, both connected through the pins of the latter.

Next, we present in a detailed way the improvements that we want to introduce in the control system:

- Remote switching on and off of 4 devices by means of relays.

- Remote switching on and off of the Raspberry Pi.
- Voltage conversion from 48V to 5V to supply the control system and devices.
- Power supply to 4 devices.
- Raspberry Pi power supply.
- PCB width same as Raspberry Pi.
- Use of DIP encapsulation except for LED and resistors with SMD encapsulation.
- Connectors for powering devices.
- Connectors for communication protocols
- Connectors for the 40-pin GPIO set.
- Implementation of SPI/USB-UART/I2C communication.

In the first part of the document, there is a brief description of what a Zybo is and what a Raspberry Pi is, its main features such as the power required for its operation or the hardware they implement, and the models used.

Subsequently, the design of each part of the control system board is carried out through the OrCAD Capture CIS circuit diagram program. The structure of the board is divided into functional blocks: connection and conversion block, control block, connection block and, finally, communication block.

The conversation block includes converters that transform the power supply voltaje from 48V to 5V so as not to damage the circuit. In the control block there are implemented relays that are controlled by a GPIO pin of the Raspberry Pi each. To protect the latter against current peaks produced by the switching of the relays.

The connection block of the devices and the Raspberry Pi contains the terminals where the power cables of the Zybo are connected, and a connector of 2x20 pins where the Raspberry Pi is connected through which it obtains the power for its operation. It is possible to add, a connector of two pins where it is the ground of the control system. And lastly, the communication block where the necessary connectors are implemented for communication through the established protocols. This block contains four three-pin connectors (SCL, GND, SDA), two five-pin connectors (GND, SSX, CLK, MISO, MOSI) and a two-pin connector (TX and RX).

At the end of this part, a series of virtual tests are carried out on the control block of the circuit designed through the Multisim program. They are divided in two phases, the first one simulates a single control block, and in the second phase the four control blocks simultaneously.

The previous circuit must then be implemented on a printed circuit board (PCB). In the design of this board first the dimensions are established and then the "footprint" of the components are distributed (with the diameter of the appropriate pads) according to the division of blocks that has been made in the schematic. After the distribution, the width of the tracks is designed according to the maximum current to be driven and the pads, then with them the connection between components is made through the pads. It should be added that, due to the impossibility of making the design on one side only, the

double-sided design is carried out. For this reason tracks are used to change from one side to the other.

Once the design of the PCB is finished, it is manufactured through a machine that prints on copper the established tracks and pads. Then, the pads of the component are drilled with a drill with a diameter two times larger than the terminal of the component. Finally, the components are soldered into place and the tracks are tinned with a cable through.

At the end of this part, two types of tests are carried out: connection tests and functional tests.

The connection tests are carried out with a voltmeter in driving mode to check that there are no short or badly welded components.

The function tests are divided into functional blocks. In the connection and conversión block where the check of whether the converter is capable of providing enough power to run the four Zybo, the check of whether the other converter is capable of providing enough power to run the Raspberry Pi and check the operation of the remote control of the converters. In the control block where the operation of the individual relay-optocoupler set is checked. The rest of the blocks are not tested because they are connectors and do not have any functionality.

In the last part of the document, the part related to communications between the Raspberry Pi and the relevant devices is designed. In this part, the interfaces of each protocol are implemented in both ends of the communication. On the Raspberry Pi side, the programming language C and its GPIO libraries are used for the interface development, while on the Zybo side, the interface is implemented through the VHDL language.

One of the implemented interfaces is the serial protocol USB-UART where the communication is done through a USB cable. The Raspberry Pi part has to be designed and the Zybo part is provided by the university. This communication is more complex than the subsequent ones, since the behavior is that of the collection of data in frame format, its printing on screen and storage in logs per device. In addition, each time an error is detected, the Zybo is restarted through the GPIO pins that control the relays. Another of the implemented interfaces is the SPI protocol where the communication is made through five cables (MOSI, MISO, SCLK, SSX and GND). This communication consists in sending numbers from 1 to 9 to the Zybo and that the Zybo returns them where it is verified that the sent bytes coincide with those received. It has implemented a reset signal that restarts the interface.

The last of the implemented interfaces is the I2C protocol where the communication is done through five cables (SDA, SCL, and GND). This communication consists in sending numbers from 1 to 9 to the Zybo and that the Zybo returns them where it is verified that the sent bytes coincide with the received ones, as it happens in the SPI communication. Another extra functionality is the configuration of the address of the devices through the "switch" of the Zybo. It has implemented a reset signal that restarts the interface.

As in the previous parts, a series of tests are carried out that demonstrate the operation of each block of the project and the complete flow that is carried out in the experiment.

The tests that are performed are: checking the control of the relays without the Zybo connected, checking the control of the relays with the Zybo connected, checking the control of the converters, checking the serial communication, checking the SPI communication through the connectors and checking the I2C communication through the connectors.

At the end of the document, the final justifications of the project are added. A list of the budget associated with the hardware design, the components used, the software design, the devices involved and the voltage source.

It details the strengths and weaknesses of this new implementation, and a series of proposals to improve these weaknesses in the future.

ANEXO B. CONVERTIDOR DE TENSIÓN DC-DC

Un convertidor de tensión es un componente electrónico que transforma una fuente de tensión de corriente continua de un valor de voltaje a otro diferente. Existen varios tipos de convertidores: aquellos que reducen la tensión de salida con respecto a la de entrada, aquellos que la elevan la salida y aquellos que tienen ambas funciones. El modelo de convertidor elegido para el circuito de la PCB es el Meanwell SKMW30G-05.

Características

SPECIFICATION				
INPUT	VOLTAGE RANGE	F: 9~36Vdc , G: 18~75Vdc		
	SURGE VOLTAGE (100ms max.)	24Vin models : 50Vdc, 48Vin models : 100Vdc		
	FILTER	Pi type		
	PROTECTION	Fuse recommended. 24Vin models: 6A delay time Type, 48Vin models: 3A delay time Type		
	INTERNAL POWER DISSIPATION	500mW		
OUTPUT	VOLTAGE ACCURACY	±1.5%		
	RATED POWER	30W		
	RIPPLE & NOISE Note.2	3.3/5Vout models: 75mVp-p, other models:100mVp-p		
	LINE REGULATION Note.3	±0.2%		
	LOAD REGULATION Note.4	Single output models: ±0.2%, Dual output models: ±1%		
	SWITCHING FREQUENCY (Typ.)	3.3/5Vout models: 270KHz, other models: 330KHz		
	EXTERNAL TRIM ADJ. RANGE (Typ.)	±10% (Single output model only)		
PROTECTION	SHORT CIRCUIT	Protection type : Continuous, automatic recovery		
	OVERLOAD	110 ~ 170% rated output power		
		Protection type : Recovers automatically after fault condition is removed		
	OVER VOLTAGE	Protection type : Clamp by diode		
	OVER TEMPERATURE	Shut down o/p voltage, recovers automatically after temperature goes down		
	UNDER VOLTAGE LOCKOUT	Start-up voltage	24Vin (F-type): 8.8Vdc, 48Vin (G-type): 17Vdc	
		Shutdown voltage	24Vin (F-type): 8Vdc, 48Vin (G-type): 16Vdc	
FUNCTION	REMOTE CONTROL	Power ON: R.C. ~ -Vin >3.5~75Vdc or open circuit ; Power OFF: R.C. ~ -Vin <1.2Vdc or short		
ENVIRONMENT	COOLING	Free-air convection		
	WORKING TEMP.	-40 ~ +85℃ (Refer to "Derating Curve")		
	CASE TEMPERATURE	+105℃ max.		
	WORKING HUMIDITY	20% ~ 90% RH non-condensing		
	STORAGE TEMP., HUMIDITY	-55 ~ +125℃, 10 ~ 95% RH non-condensing		
	TEMP. COEFFICIENT	0.03% / ℃ (0 ~ 60℃)		
	SOLDERING TEMPERATURE	1.5mm from case of 1 ~ 3sec./260℃ max.		
	VIBRATION	10 ~ 500Hz, 2G 10min./1cycle, period for 60min. each along X, Y, Z axes		
SAFETY & EMC (Note.5)	WITHSTAND VOLTAGE	I/P-O/P:1.5KVDC		
	ISOLATION RESISTANCE	I/P-O/P:100M Ohms / 500VDC / 25℃ / 70% RH		
	ISOLATION CAPACITANCE (Typ.)	1500pF		
	EMC EMISSION	**Parameter**	**Standard**	**Test Level / Note**
		Conducted	EN55032(CISPR32)	N/A
		Radiated	EN55032(CISPR32)	Class A
	EMC IMMUNITY	**Parameter**	**Standard**	**Test Level / Note**
		ESD	EN61000-4-2	Level 2, ±8KV air, ±4KV contact
		Radiated Susceptibility	EN61000-4-3	Level 2, 3V/m
		EFT/Burest	EN61000-4-4	Level 1, 0.5KV
		Surge	EN61000-4-5	Level 1, 0.5KV Line-Line
		Conducted	EN61000-4-6	Level 2, 3V(e.m.f.)
		Magnetic Field	EN61000-4-8	Level 2, 3A/m
OTHERS	MTBF	3.3/5Vout models: 860Khrs, Other models: 1170Khrs MIL-HDBK-217F(25℃)		
	DIMENSION (L*W*H)	25.4*25.4*10.2mm (1*1*0.4 inch)		
	CASE MATERIAL	Black coated copper with non-conductive base		
	PACKING	18g		

Fig. B.1. Hoja de características del SKMW30G-05

Símbolo

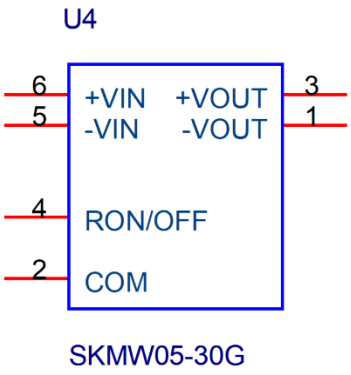


Fig. B.2. Símbolo esquemático del convertidor

Huella

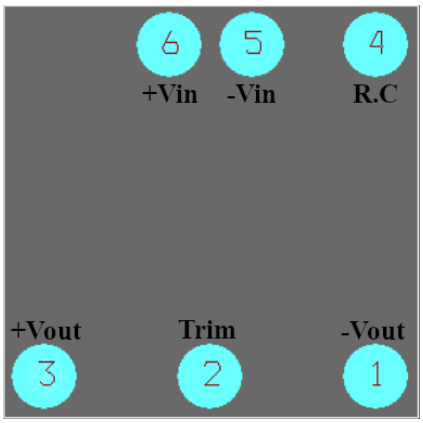


Fig. B.3. Huella del convertidor

ANEXO C. OPTOACOPLADOR

El optoacoplador es un componente electrónico que se encarga de aislar de manera óptica una parte del circuito, lo que supone a nivel físico estar separados por una resistencia del orden de MΩ, comportándose como un conmutador óptico. Este se compone de un diodo LED y un fototransistor. El modelo escogido para implementar en la PCB es el TPC817C.

Características

ABSOLUTE MAXIMUM RATINGS (T_A = 25°C unless otherwise noted)

PARAMETER		SYMBOL	PART NUMBER	UNIT
Input	Forward current	I_F	50	mA
	Reverse voltage	V_R	6	V
	Power dissipation	P	70	mW
Output	Collector-emitter voltage	V_{CEO}	80	V
	Emitter-collector voltage	V_{ECO}	6	V
	Collector current	I_C	50	mA
	Collector power dissipation	P_C	150	mW
Total power dissipation		P_{tot}	200	mW
Isolation voltage		V_{iso}	5000	Vrms
Rated impulse isolation voltage		V_{IOTM}	6000	V
Rated repetitive peak isolation voltage		V_{IORM}	630	V
Operating temperature		T_{opr}	-40 to +100	°C
Storage temperature		T_{stg}	-55 to +125	°C
Soldering temperature		T_{sol}	260	°C

ELECTRICAL SPECIFICATIONS (T_A = 25°C unless otherwise noted)

PARAMETER			CONDITIONS	SYMBOL	MIN	TYP	MAX	UNIT
Input	Forward voltage		I_F=20mA	V_F		1.2	1.4	V
	Reverse current		V_R=4V	I_R			10	μA
	Terminal capacitance		V=0, f=1kHz	C_t		30	250	pF
Output	Collector dark current		V_{CE}=20V, I_F=0	I_{CEO}			10^{-7}	A
	Collector-emitter breakdown voltage		I_C=0.1mA, I_F=0	BV_{CEO}	80			V
	Emitter-collector breakdown voltage		I_E=10μA, I_F=0	BV_{ECO}	6			V
Transfer Characteristics	Collector current		I_F=5mA, V_{CE}=5V	IC	2.5		30	mA
	Current transfer ration(Note 1)			CTR	80		600	%
	Collector-emitter saturation voltage		I_F=20mA, I_C=1mA	$V_{CE(sat)}$		0.1	0.2	V
	Isolation resistance		DC500V, 40 to 60%RH	R_{ISO}	$5x10^{10}$	10^{11}		Ω
	Floating capacitance		V=0, f=1MHz	C_f		0.6	1.0	pF
	Cut-off frequency		V_{CE}=5V, I_C=2mA, R_L=100Ω, -3dB	f_c		80		KHz
	Response time	Rise time	V_{CE}=2V, I_C=2mA, R_L=100Ω	t_r		4	18	μs
		Fall time		t_f		3	18	μs

Fig. C.1. Hoja de características del TPC817C

Símbolo

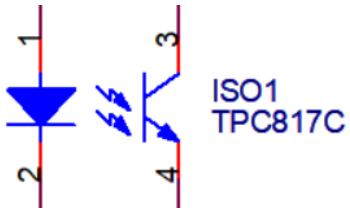


Fig. C.2. Símbolo esquemático del optoacoplador

Huella

Fig. C.3. Huella del optoacoplador

ANEXO D. TRANSISTOR

El transistor es un componente electrónico que se encargar de amplificar a la salida una pequeña corriente que se presenta a su entrada, o de la conmutación de señales eléctricas. Está formado por dos uniones NP o PN separadas entre sí, formando tres regiones diferencia das: colector, base y emisor. Existen dos tipos: bipolares (BJT) y de efecto campo (JFET), y dentro de estos en función de su tipo de unión pueden ser NPN o PNP. El modelo escogido para implementar en la PCB es el BC547B.

Características

Absolute Maximum Ratings

Stresses exceeding the absolute maximum ratings may damage the device. The device may not function or be operable above the recommended operating conditions and stressing the parts to these levels is not recommended. In addition, extended exposure to stresses above the recommended operating conditions may affect device reliability. The absolute maximum ratings are stress ratings only. Values are at T_A = 25°C unless otherwise noted.

Symbol	Parameter		Value	Unit
V_{CBO}	Collector-Base Voltage	BC546	80	V
		BC547 / BC550	50	
		BC548 / BC549	30	
V_{CEO}	Collector-Emitter Voltage	BC546	65	V
		BC547 / BC550	45	
		BC548 / BC549	30	
V_{EBO}	Emitter-Base Voltage	BC546 / BC547	6	V
		BC548 / BC549 / BC550	5	
I_C	Collector Current (DC)		100	mA
P_C	Collector Power Dissipation		500	mW
T_J	Junction Temperature		150	°C
T_{STG}	Storage Temperature Range		-65 to +150	°C

Electrical Characteristics

Values are at T_A = 25°C unless otherwise noted.

Symbol	Parameter		Conditions	Min.	Typ.	Max.	Unit
I_{CBO}	Collector Cut-Off Current		V_{CB} = 30 V, I_E = 0			15	nA
h_{FE}	DC Current Gain		V_{CE} = 5 V, I_C = 2 mA	110		800	
V_{CE}(sat)	Collector-Emitter Saturation Voltage		I_C = 10 mA, I_B = 0.5 mA		90	250	mV
			I_C = 100 mA, I_B = 5 mA		250	600	
V_{BE}(sat)	Base-Emitter Saturation Voltage		I_C = 10 mA, I_B = 0.5 mA		700		mV
			I_C = 100 mA, I_B = 5 mA		900		
V_{BE}(on)	Base-Emitter On Voltage		V_{CE} = 5 V, I_C = 2 mA	580	660	700	mV
			V_{CE} = 5 V, I_C = 10 mA			720	
f_T	Current Gain Bandwidth Product		V_{CE} = 5 V, I_C = 10 mA, f = 100 MHz		300		MHz
C_{ob}	Output Capacitance		V_{CB} = 10 V, I_E = 0, f = 1 MHz		3.5	6.0	pF
C_{ib}	Input Capacitance		V_{EB} = 0.5 V, I_C = 0, f = 1 MHz		9		pF
NF	Noise Figure	BC546 / BC547 / BC548	V_{CE} = 5 V, I_C = 200 µA, f = 1 kHz, R_G = 2 kΩ		2.0	10.0	dB
		BC549 / BC550			1.2	4.0	
		BC549	V_{CE} = 5 V, I_C = 200 µA, R_G = 2 kΩ, f = 30 to 15000 MHz		1.4	4.0	
		BC550			1.4	3.0	

h_{FE} Classification

Classification	A	B	C
h_{FE}	110 ~ 220	200 ~ 450	420 ~ 800

Fig. D.1. Hoja de características del BC547B

Símbolo

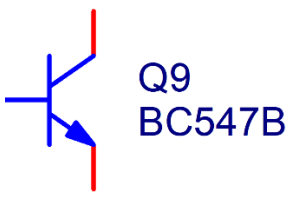


Fig. D.2. Símbolo esquemático del transistor

Huella

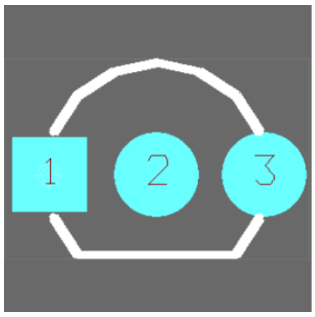


Fig. D.3. Huella del transistor

ANEXO E. LED

El LED (light-emitter diode) es un componente semiconductor que al igual que los diodos solo deja pasar la corriente en un sentido, pero en este caso el paso de este excita el diodo de tal manera que comienza a emitir luz. El modelo escogido para la implementación del LED es el modelo APHHS1005LQBC/D-V.

Características

Kingbright

APHHS1005LQBC/D-V

ELECTRICAL / OPTICAL CHARACTERISTICS at T_A=25°C

Parameter	Symbol	Emitting Color	Value			Unit
			Min.	Typ.	Max.	
Wavelength at Peak Emission I_F = 2mA	λ_{peak}	Blue	-	460	-	nm
Dominant Wavelength I_F = 2mA	λ_{dom} [1]	Blue	-	465	-	nm
Spectral Bandwidth at 50% Φ REL MAX I_F = 2mA	Δλ	Blue	-	25	-	nm
Capacitance	C	Blue	-	100	-	pF
Forward Voltage I_F = 2mA	V_F [2]	Blue	2.2	2.65	3	V
Reverse Current (V_R = 5V)	I_R	Blue	-	-	50	μA

Notes:
1. *The dominant wavelength (λd) above is the setup value of the sorting machine. (Tolerance λd : ±1nm.)*
2. *Forward voltage: ±0.1V.*
3. *Wavelength value is traceable to CIE127-2007 standards.*
4. *Excess driving current and / or operating temperature higher than recommended conditions may result in severe light degradation or premature failure.*

ABSOLUTE MAXIMUM RATINGS at T_A=25°C

Parameter	Symbol	Value	Unit
Power Dissipation	P_D	120	mW
Reverse Voltage	V_R	5	V
Junction Temperature	T_j	115	°C
Operating Temperature	T_{op}	-40 to +85	°C
Storage Temperature	T_{stg}	-40 to +85	°C
DC Forward Current	I_F	30	mA
Peak Forward Current	I_{FM} [1]	150	mA
Electrostatic Discharge Threshold (HBM)	-	250	V

Fig. E.1. Hoja de las características del APHHS1005LQBC

Símbolo

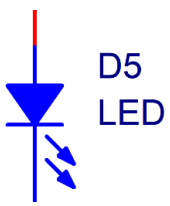


Fig. E.24. Símbolo esquemático del LED

Huella

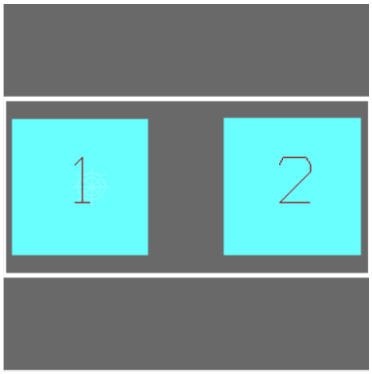


Fig. E.35. Huella del led

ANEXO F. RELÉ

El relé es un componente electromagnético que funciona a modo de interruptor, abriendo o cerrando el paso de la corriente en una parte del circuito. No es un interruptor manual, sino un interruptor que se activa eléctricamente. El modelo escogido para la implementación del relé es un relé electromecánico cuyo modelo es Finder 40.51.9006.0000.

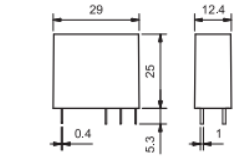
Características

finder — 40 Series - Miniature PCB/Plug-in relays 8 - 10 - 16 A

Features

1 & 2 Pole relay range
40.31 - 1 Pole 10 A (3.5 mm pin pitch)
40.51 - 1 Pole 10 A (5 mm pin pitch)
40.52 - 2 Pole 8 A (5 mm pin pitch)
PCB mount
- direct or via PCB socket
35 mm rail mount
- via screw and screwless sockets

- DC coils (standard or sensitive) & AC coils
- Cadmium Free contact material
- 8 mm, 6 kV (1.2/50 µs) isolation, coil-contacts
- UL Listing (certain relay/socket combinations)
- Flux proof: RT II standard, (RT III option)
- 95 series sockets
- Coil EMC suppression
- Timer accessories 86 series



For UL Horsepower and Pilot Duty ratings See "General technical information" page V

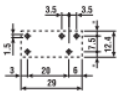
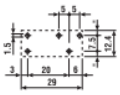
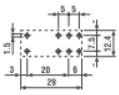
	40.31	40.51	40.52
			
	• 3.5 mm contact pin pitch • 1 Pole 10 A • PCB or 95 series sockets	• 5 mm contact pin pitch • 1 Pole 10 A • PCB or 95 series sockets	• 5 mm contact pin pitch • 2 Pole 8 A • PCB or 95 series sockets
			
			
	Copper side view	Copper side view	Copper side view
Contact specification			
Contact configuration	1 CO (SPDT)	1 CO (SPDT)	2 CO (DPDT)
Rated current/Maximum peak current A	10/20	10/20	8/15
Rated voltage/Maximum switching voltage V AC	250/400	250/400	250/400
Rated load AC1 VA	2,500	2,500	2,000
Rated load AC15 (230 V AC) VA	500	500	400
Single phase motor rating (230 V AC) kW	0.37	0.37	0.3
Breaking capacity DC1: 30/110/220 V A	10/0.3/0.12	10/0.3/0.12	8/0.3/0.12
Minimum switching load mW (V/mA)	300 (5/5)	300 (5/5)	300 (5/5)
Standard contact material	AgNi	AgNi	AgNi
Coil specification			
Nominal voltage (U_N) V AC (50/60 Hz)	6 - 12 - 24 - 48 - 60 - 110 - 120 - 230 - 240		
V DC	5 - 6 - 7 - 9 - 12 - 14 - 18 - 21 - 24 - 28 - 36 - 48 - 60 - 90 - 110 - 125		
Rated power AC/DC/sens. DC VA (50 Hz)/W/W	1.2/0.65/0.5	1.2/0.65/0.5	1.2/0.65/0.5
Operating range AC	(0.8...1.1)U_N	(0.8...1.1)U_N	(0.8...1.1)U_N
DC/sens. DC	(0.73...1.5)U_N/(0.73...1.75)U_N	(0.73...1.5)U_N/(0.73...1.75)U_N	(0.73...1.5)U_N/(0.73...1.75)U_N
Holding voltage AC/DC	0.8 U_N /0.4 U_N	0.8 U_N /0.4 U_N	0.8 U_N /0.4 U_N
Must drop-out voltage AC/DC	0.2 U_N /0.1 U_N	0.2 U_N /0.1 U_N	0.2 U_N /0.1 U_N
Technical data			
Mechanical life AC/DC cycles	10 · 10^6/20 · 10^6	10 · 10^6/20 · 10^6	10 · 10^6/20 · 10^6
Electrical life at rated load AC1 cycles	200 · 10^3	200 · 10^3	100 · 10^3
Operate/release time ms	7/3 - (12/4 sensitive)	7/3 - (12/4 sensitive)	7/3 - (12/4 sensitive)
Insulation between coil and contacts (1.2/50 µs) kV	6 (8 mm)	6 (8 mm)	6 (8 mm)
Dielectric strength between open contacts V AC	1,000	1,000	1,000
Ambient temperature range °C	–40...+85	–40...+85	–40...+85
Environmental protection	RT II**	RT II**	RT II**

Fig. F.1. Hoja de características del Finder 40.51

Símbolo

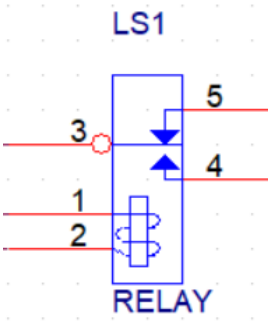


Fig. F.2. Símbolo esquemático del relé

Huella

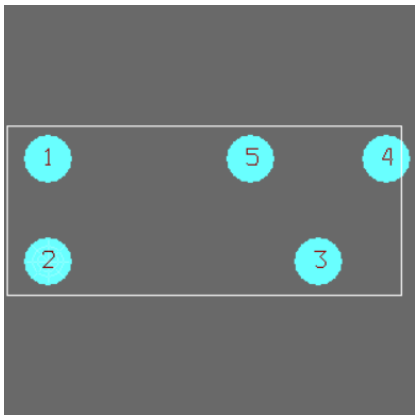


Fig. F.3. Huella del relé

ANEXO G. CONECTORES DE PINES

En este caso las tiras de pines son conectores macho con una separación entre pines de 2.54mm. Están implementados en conectores de 2, 3, 5 y 2x20 pines.

Símbolo

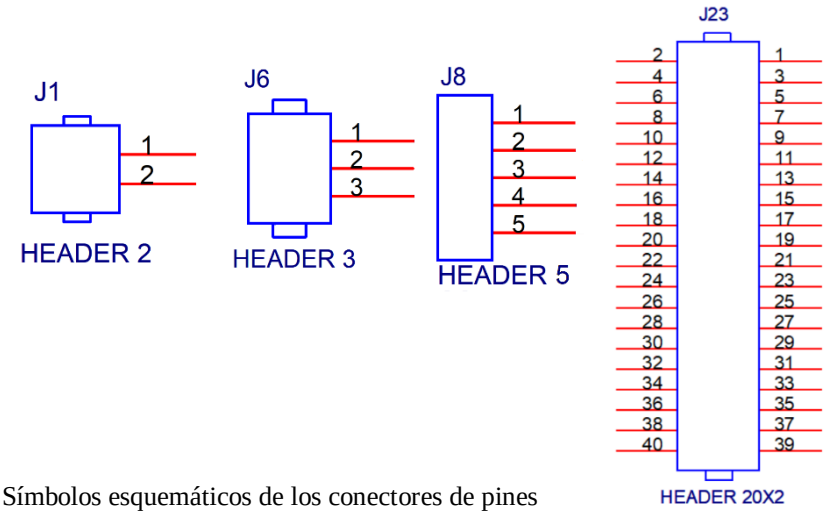


Fig. G.1. Símbolos esquemáticos de los conectores de pines

Huella

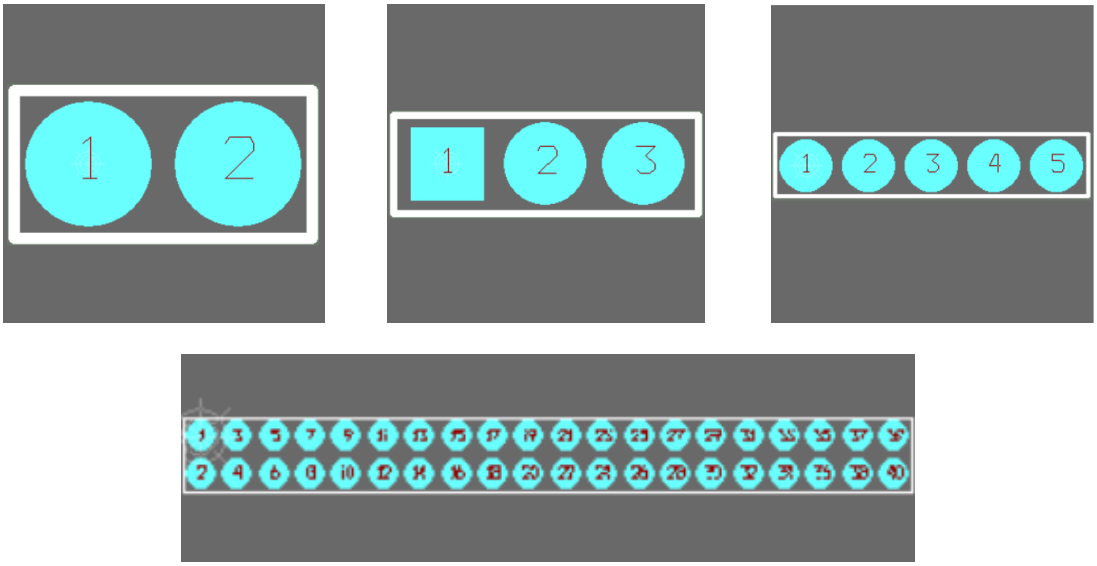


Fig. G.2. Huellas de los conectores de pines

ANEXO H. CLEMA

Las clemas implementadas en este proyecto son de doble polo por puerto (2 o 3) y se introducen mediante la presión de una pestaña lo que facilita la conexión y desconexión de componentes.

Símbolo

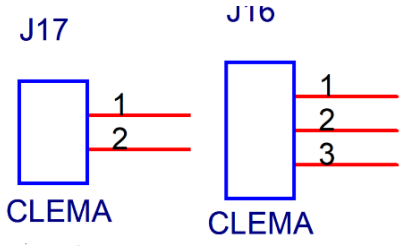


Fig. H.1. Símbolos esquemáticos de las clemas

Huella

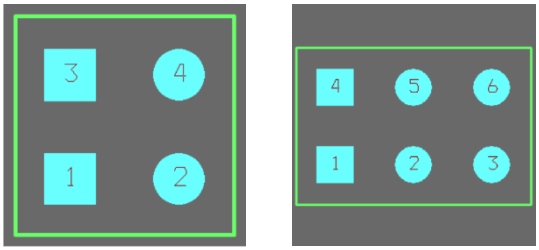


Fig. H.2. Huellas de las clemas

ANEXO I. MULTISIM

“Multisim es un software estándar en la industria para diseño de circuitos y simulación SPICE para electrónica de potencia, analógica y digital en la educación y la investigación”.[41]

“Multisim integra simulación SPICE estándar en la industria con un entorno esquemático interactivo para visualizar y analizar al instante el comportamiento de los circuitos electrónicos. Multisim tiene una interfaz intuitiva que ayuda a los profesores a reforzar la teoría de circuitos y a mejorar la teoría en todo el plan de estudios de ingeniería. Investigadores y diseñadores utilizan Multisim para reducir las iteraciones de prototipos PCB y ahorrar costos de desarrollo, añadiendo simulación potente de circuitos y análisis al flujo de diseño”.[41]

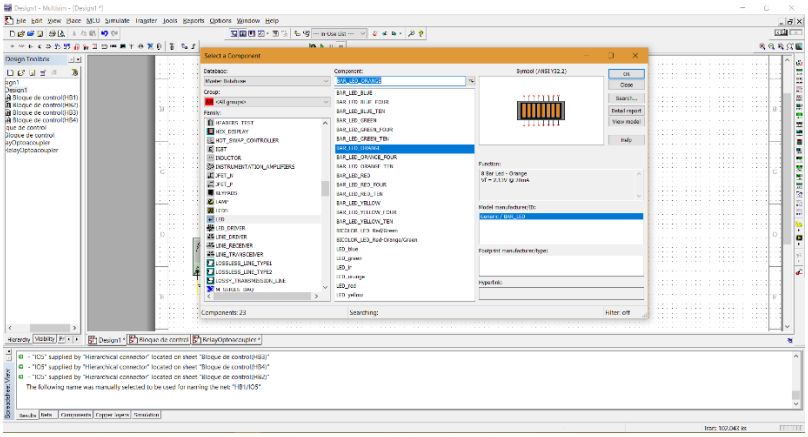


Fig. I1. Multisim[41]

ANEXO J. ORCAD

"ORCAD 10.3 es un programa ampliamente utilizado para el diseño de circuitos electrónicos. Consta de dos bloques básicos: una herramienta para la simulación del comportamiento de circuitos electrónicos (PSPICE) y una herramienta para el diseño de placas de circuito impreso y PCB(Layout). Como paso previo para la simulación del circuito y el diseño del PCB es necesario realizar la captura del esquema del circuito que se quiere analizar."[42]

El programa Capture CIS es el que se utiliza para llevar a cabo el diseño esquemático del circuito de la placa del sistema de control, mientras que el Layout Plus se utiliza para implementar el diseño esquemático que se ha elaborado en el anterior programa en una PCB.

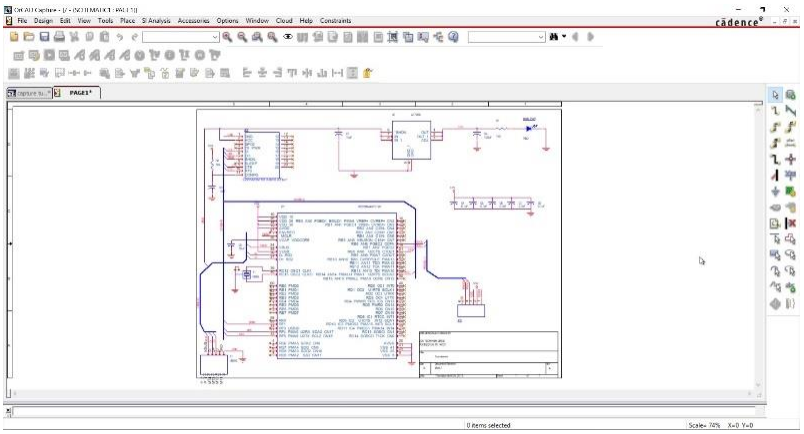


Fig. J.1. Capture CIS[43]

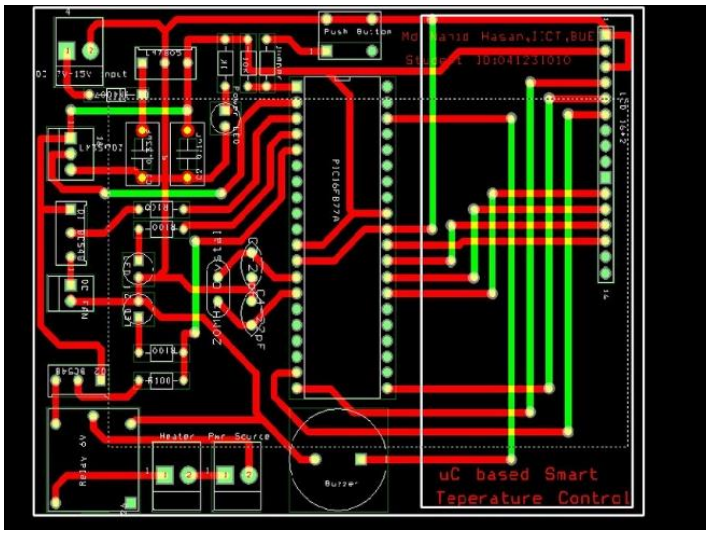


Fig. J.2. Layout Plus[44]

ANEXO K. PCB TRACE WIDTH CALCULATOR

Es una herramienta online para calcular los anchos de las pistas para una PCB.

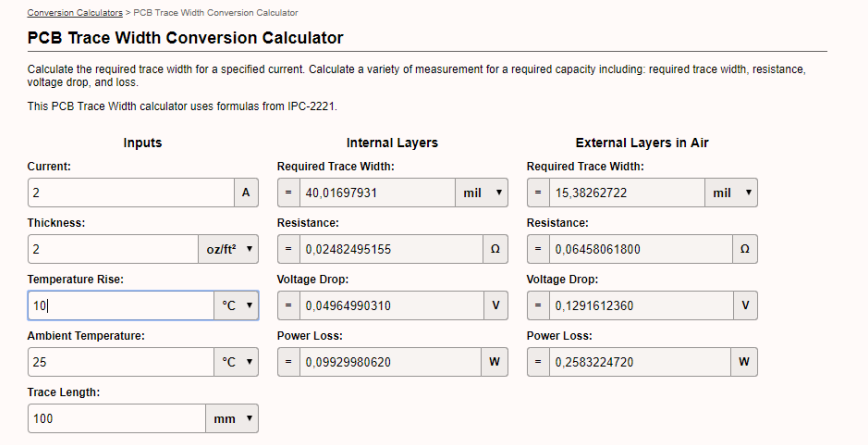


Fig. K.1. Ejemplo de la herramienta PCB Trace Width Calculator

ANEXO L. PROGRAMAR UNA ZYBO

Antes de nada, es necesario conectar la Zybo al ordenador por medio de un cable micro USB a USB. Hay que asegurarse de que el jumper de la Zybo esté en modo QSPI, como se ha mencionado en su apartado específico, en este modo no se borra la implementación programada en la placa.

En primer lugar, se abre la aplicación VIVADO 2015.2, y se selecciona la opción "Open hardware manager". Después de que es reconocida la Zybo. Aparece un panel que aparece en la parte izquierda de la **Figura**, damos clic derecho a "xc7z010_1" y después a "Add Configuration Memory Device". Se nos muestra una ventana emergente donde encontramos una barra de búsqueda (2) en ella introducimos el nombre de la memoria donde queremos que se instale el programa, en este caso, la memoria es "S25FL128S-3.3V-qspi-x4-single" y la seleccionamos (3).

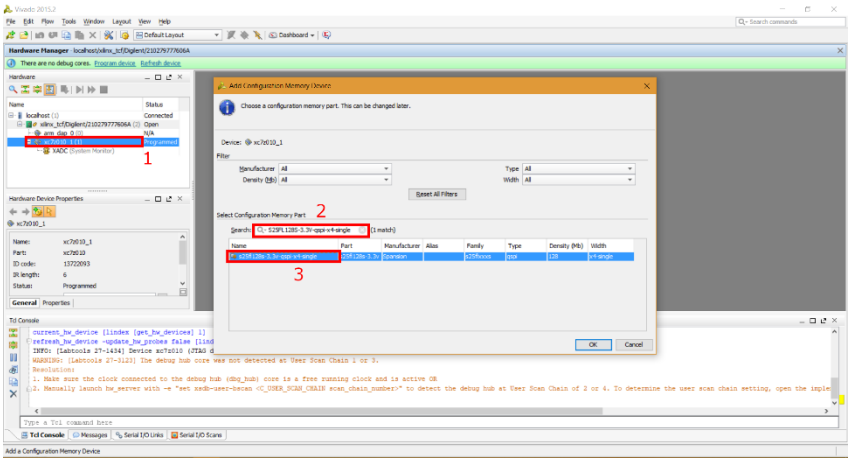


Fig. L.1. Instalación del programa en la Zybo con Vivado 2015.2

Después, nos aparece otra ventana en donde nos encontraremos con un apartado llamado "Configuration file". En este apartado se introduce el binario que contiene el programa y el resto de las opciones se dejan por defecto. Le damos al botón "Ok" y empieza el proceso de instalación en la Zybo.

Si la instalación ha sido exitosa, se enciende el LED "DONE" de la Zybo.

Existe otro método para programar una Zybo. Para ello se configura el Jumper en modo 'JTAG'.

Esta programación se utiliza para instalar programas desde un archivo .bit a través de la interfaz JTAG. La instalación de este programa se hace en una memoria volátil, es decir, si se interrumpe la corriente se borra.

El proceso de instalación comienza cuando se pulsa la opción "Program device". En la ventana siguiente seleccionamos el archivo .bit que queremos instalar.

Al igual que en el caso anterior, si la instalación ha sido exitosa, se enciende el LED "DONE" de la Zybo.

ANEXO M. PROGRAMA SPI RASPBERRY PI

spi.c

```
#include <stdio.h>
#include <bcm2835.h>
#include <errno.h>
#include <unistd.h>

#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_RESET   "\x1b[0m"

static int cs = 0;
static int pin = 1;

static void parse_opts(int argc, char *argv[])
{
        while (1) {
                int c;

                c = getopt(argc, argv, "c:p:");

                if (c == -1)
                        break;

                switch (c) {
                case 'c':
                        cs = atoi(optarg);
                        break;
                case 'p':
                        pin = atoi(optarg);
                }
        }
}
void enable_device(int pin){
        switch(pin){
                case 1:
                        bcm2835_gpio_fsel(RPI_V2_GPIO_P1_13, BCM2835_GPIO_FSEL_OUTP);
                        bcm2835_gpio_write(RPI_V2_GPIO_P1_13, LOW);
                break;

                case 2:
                        bcm2835_gpio_fsel(RPI_V2_GPIO_P1_11, BCM2835_GPIO_FSEL_OUTP);
                        bcm2835_gpio_write(RPI_V2_GPIO_P1_11, LOW);
                break;

                case 3:
                        bcm2835_gpio_fsel(RPI_V2_GPIO_P1_16, BCM2835_GPIO_FSEL_OUTP);
                        bcm2835_gpio_write(RPI_V2_GPIO_P1_16, LOW);
                break;

                case 4:
                        bcm2835_gpio_fsel(RPI_V2_GPIO_P1_31 ,
BCM2835_GPIO_FSEL_OUTP);
                        bcm2835_gpio_write(RPI_V2_GPIO_P1_31, LOW);
                break;
        }
}

int main (int argc, char *argv[]) {
```

```
        int mode = 0;
        int clock_divider = 256;
        float pi_clock_speed = 250.0f;
        float speed = 0.0;

        parse_opts(argc, argv);

    bcm2835_init();
        printf("Enabling device...\n");
        enable_device(pin);
        printf("Device enabled\n");
    if (bcm2835_spi_begin()==0){
                fprintf (stderr, "Unable to open SPI bus: %s\n", strerror (errno));
                return -1;
        }
        printf("Configuring SPI connection...\n");
    bcm2835_spi_setDataMode(mode);
        printf("Set SPI mode %d\n", mode);
    bcm2835_spi_setClockDivider(clock_divider);
        speed = pi_clock_speed/clock_divider;
        printf("Set SPI speed %f MHz\n", speed);
    bcm2835_spi_chipSelect(cs);
        printf("Set SPI channel to %d\n", cs);
        printf("SPI connection configured\n");

    char tdata[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 0};
    char rdata[10];
        printf("Testing SPI connection...\n");
    bcm2835_spi_transfernb(tdata, rdata, 10);
        printf("Enviado: %d %d %d %d %d %d %d %d %d %d\n", tdata[0], tdata[1], tdata[2], tdata[3],
tdata[4], tdata[5], tdata[6], tdata[7], tdata[8]);
    printf("Recibido: %d %d %d %d %d %d %d %d %d %d\n", rdata[1], rdata[2], rdata[3], rdata[4], rdata[5],
rdata[6], rdata[7], rdata[8], rdata[9]);//Delay for 1 byte
        if(rdata[1]==1 & rdata[2]==2 & rdata[3]==3 & rdata[4]==4 & rdata[5]==5 & rdata[6]==6 &
rdata[7]==7 & rdata[8]==8 & rdata[9]==9){
                printf("Validation: "ANSI_COLOR_GREEN"OK\n"ANSI_COLOR_RESET);
        }else{
                printf("Validation: "ANSI_COLOR_RED"ERROR\n"ANSI_COLOR_RESET);
                printf("Test SPI connection:
"ANSI_COLOR_RED"ERROR\n"ANSI_COLOR_RESET);
                bcm2835_spi_end();
                bcm2835_close();
                return -1;
        }
        printf("Test SPI connection: "ANSI_COLOR_GREEN"OK\n"ANSI_COLOR_RESET);
    bcm2835_spi_end();
    bcm2835_close();
    return 0;
}
```

ANEXO N. INTEFAZ SPI ZYBO

spi-slave.vhdl

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

entity SPI is
generic(width: INTEGER := 8;
width_counter: INTEGER := 3);
   Port ( CLK : in STD_LOGIC;
          RST : in STD_LOGIC;
          SCLK : in STD_LOGIC;
          MOSI : in STD_LOGIC;
          SS_N : in STD_LOGIC;
          MISO : out STD_LOGIC);
end SPI;

architecture Behavioral of SPI is

signal SCKL_reg:           std_logic_vector(2 DOWNTO 0);
signal SCLK_risingedge:    std_logic;
signal SCLK_fallingedge:   std_logic;
signal SS_N_reg:           std_logic_vector(2 DOWNTO 0);
signal SS_N_active:        std_logic;
signal SS_N_startmessage:  std_logic;
signal SS_N_endmessage:    std_logic;
signal MOSI_reg:           std_logic_vector(1 DOWNTO 0);
signal MOSI_data:          std_logic;
signal bitcount:           unsigned(width_counter-1 DOWNTO 0);
signal byte_data_received: std_logic_vector(width-1 DOWNTO 0);
signal bit_cnt_max:        std_logic;

begin

   process(clk)
   begin
      if (rising_edge(CLK)) then
            if (RST = '1') then
               SCKL_reg <= "000";
            else
               SCKL_reg <= SCKL_reg(1 DOWNTO 0) & SCLK;
            end if;
      end if;
   end process;

   SCLK_risingedge <= '1' WHEN (SCKL_reg(2 DOWNTO 1) = "01") else '0';
   SCLK_fallingedge <= '1' WHEN (SCKL_reg(2 DOWNTO 1) = "10") else '0';

   process(CLK)
   begin
      if (rising_edge(CLK)) then
               if (RST = '1') then
                  SS_N_reg <= "000";
               else
                  SS_N_reg <= SS_N_reg(1 DOWNTO 0) & SS_N;
```

```
            end if;
        end if;
    end process;

    SS_N_active <= NOT SS_N_reg(1);
    SS_N_startmessage <= '1' WHEN (SS_N_reg(2 DOWNTO 1)="10") else '0';
    SS_N_endmessage <= '1' WHEN (SS_N_reg(2 DOWNTO 1)="01") else '0';

    process(CLK)
    begin
      if (rising_edge(CLK)) then
            if (RST = '1') then
              MOSI_reg <= "00";
            else
              MOSI_reg <= MOSI_reg(0) & MOSI;
            end if;
        end if;
    end process;

    MOSI_data <= MOSI_reg(1);

    process(CLK)
    begin
      if (rising_edge(CLK)) then
        if (NOT SS_N_active = '1' OR RST='1') then
          bitcount <= to_unsigned(0,width_counter);
          byte_data_received <= std_logic_vector(to_unsigned(0,width));
        else
          if(SCLK_risingedge = '1') then
            if(bitcount = to_unsigned(width-1,width_counter)) then
                                                    bitcount <= to_unsigned(0,width_counter);
            else
              bitcount <= bitcount + 1;
            end if;
            byte_data_received <= byte_data_received(width-2 DOWNTO 0) & MOSI_data; --Out of
else for last bit shift
          end if;
        end if;
      end if;
    end process;

    bit_cnt_max <= '1' when (bitcount=to_unsigned(width-1,width_counter)) else '0';

    process(CLK)
      begin
        if(rising_edge(CLK)) then
          if(NOT SS_N_active = '1' OR RST='1') then
            MISO <= 'Z';
          elsif(SS_N_active = '1' AND SCLK_fallingedge='1') then
            MISO <= byte_data_received(7);
          end if;
        end if;
    end process;

end Behavioral;
```

ANEXO O. PROGRAMA I2C RASPBERRY PI

i2c.c

```
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <time.h>
#include <unistd.h>

#define ANSI_COLOR_RED     "\x1b[31m"
#define ANSI_COLOR_GREEN   "\x1b[32m"
#define ANSI_COLOR_RESET   "\x1b[0m"
#define ROW      4
#define COL      4

struct Devices {
        int *fd;
        int *address;
        int **test;
};

void gpio_activate(){

        pinMode(2,OUTPUT);
        digitalWrite(2,LOW);

        pinMode(0,OUTPUT);
        digitalWrite(0,LOW);

        pinMode(4,OUTPUT);
        digitalWrite(4,LOW);

        pinMode(22,OUTPUT);
        digitalWrite(22,LOW);
}
void sent_data(struct Devices *available_devices, int i){
        int data;
        int length = sizeof(available_devices->fd);
        printf("\n");
        for(int j = 0; j < length ;j++){
                wiringPiI2CWrite (available_devices->fd[j], i);
                printf("Sent to Zybo %#04x: %d                ", available_devices->address[j], i);
        }
}

int read_data(struct Devices *available_devices, int i, int **test){
        int data = 0;
        int length = sizeof(available_devices->fd);
        printf("\n");
        for(int j = 0; j < length ;j++){
                data = wiringPiI2CRead(available_devices->fd[j]);
                if(data==-1){
                        printf("Received from Zybo %#04x: "ANSI_COLOR_RED"NOT
CONNECTED              "ANSI_COLOR_RESET, j);
                }else{
                        printf("Received from Zybo %#04x: %d       ", available_devices-
>address[j],data);
                }
                test_data(j,i,data, test);
```

```
        }
        return test;
}
int test_data(int j, int i, int data, int **test){
   if(data==-1)
   {
                return -1;
   }
   else
   {
                test[j][i]=data;
   }
        return 0;
}

int main (int argc, char *argv[])
{
                struct Devices *available_devices = NULL;
                int i = 0;
                int t = 0;
      int data;
                time_t endwait;
                time_t start;
                time_t seconds = 10;
                int sent = 0;
                int retries = 0;
                int received = 0;

                if(argc < 6 && argc > 1){
                        available_devices = (struct Devices *) malloc(sizeof(struct Devices));
                        available_devices->fd = (int *)malloc((argc-1)*sizeof(int));
                        available_devices->address = (int *)malloc((argc-1)*sizeof(int));
                        available_devices->test = (int **)malloc((argc-1) * sizeof(int *));
                        for(int i=0; i < (argc-1);i++){
                                available_devices->test[i] = (int *)malloc(10 * sizeof(int));
                        }

                                for(int i=1; i < (argc);i++){
                                        int address = (int) strtol(argv[i], NULL, 0);
                                        printf("Configuring address:%#04x\n",address);
                                        available_devices->address[i-1] = address;
                                        available_devices->fd[i-1] =
wiringPiI2CSetup(available_devices->address[i-1]);
                                                                if(available_devices->fd[i]== -1){
                                                                        printf("I2C device not
configured\n");
                                                                        return -1;
                                                                }
                                }
                }else{
                        printf(ANSI_COLOR_RED"Introduce minium one address and maxium four
address\n"ANSI_COLOR_RESET);
                        return -1;
                }
                        while(i < 10){
                                sent_data(available_devices,i);
                                delay(100);
                                read_data(available_devices, i, available_devices->test);
                                i++;
        }
```

```
                        printf("\n");
                        for(int Index = 0; Index < 4; Index++){
                                printf("Check Zybo %#04x ...                  ", available_devices->address[Index]);
                                for(int Index2 = 0; Index2 < 10; Index2++) {
                                        if (available_devices->test[Index][Index2] != Index2) {
                                                printf("\nReceiving sent data of slave %#04x: "ANSI_COLOR_RED"CHECK ERROR.\n"ANSI_COLOR_RESET,available_devices->address[Index]);
                                                return -1;
                                                break;
                                        }
                                }
                        }
                printf("\nReceiving sent data of slave test: "ANSI_COLOR_GREEN"OK.\n"ANSI_COLOR_RESET);
                free(available_devices->address);
                free(available_devices->fd);
                for(int i=0; i < (argc-1);i++){
                        free(available_devices->test[i]);
                }
                free(available_devices->test);
                free(available_devices);
        return 0;
}
```

ANEXO P. INTERFAZ I2C ZYBO

i2c-salve.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity I2C is
   port (
   SCL          : inout STD_LOGIC;
   SDA          : inout STD_LOGIC;
   CLK          : in STD_LOGIC;
   SWT          : in STD_LOGIC_VECTOR(3 DOWNTO 0);
   LED          : out STD_LOGIC_VECTOR(3 DOWNTO 0));
end I2C;


architecture i2c_slave_fsm of I2C is
   --Address
   signal SLAVE_ADDR : std_logic_vector(6 downto 0):= "0000000";

   --Output signals
   signal sda_out_en      : std_logic := '0';
   signal sda_o           : std_logic := '0';
   signal rx_data_rdy_reg : std_logic          := '0';
   signal tx_byte_buf     : std_logic_vector(7 downto 0) := (others => '0');

    --Pipelined SDA and SCL signals
   signal scl_d   : std_logic := '1';
   signal scl_d2  : std_logic := '1';
   signal sda_d   : std_logic := '1';
   signal sda_d2  : std_logic := '1';

   --Strobes for SCL edges and Start/Stop bits
   signal start_strobe      : std_logic := '0';
   signal stop_strobe       : std_logic := '0';
   signal scl_rising_strobe : std_logic := '0';
   signal scl_falling_strobe : std_logic := '0';

    --I2C FSM control signals
   type state_t is (IDLE, READ_ADDRESS, SEND_ACK_1, WRITE_CMD,
                    READ_CMD, WAIT_ACK_1, WAIT_ACK_2, WAIT_STOP);

   signal state          : state_t          := IDLE;
   signal rw_command     : std_logic        := '0';
   signal bit_counter    : integer range 0 to 8 := 0;
   signal continue_read  : std_logic        := '0';

   --Signals for storing address/data from master
   signal addr_buf    : std_logic_vector(6 downto 0) := (others => '0');
   signal rx_data_buf : std_logic_vector(7 downto 0) := (others => '0');

   signal tx_byte : std_logic_vector(7 downto 0):="10000010";

begin

--I2C State Machine
i2c_fsm_p : process (clk) is
begin
   if rising_edge(clk) then
```

```
-- Default values
sda_o       <= '0';
sda_out_en  <= '0';
rx_data_rdy_reg <= '0';

case state is
   --Idle state (functionally same as STOP)
   when IDLE =>
      if start_strobe = '1' then
         state          <= READ_ADDRESS;
         bit_counter <= 0;
      end if;

   --Reading 7 bit address from master
   when READ_ADDRESS =>
      if scl_rising_strobe = '1' then
         if bit_counter < 7 then
            --Store the first 7 bits that are read into the address buffer
            bit_counter             <= bit_counter + 1;
            addr_buf(6-bit_counter) <= sda_d;
         elsif bit_counter = 7 then
            --The next bit indicates whether it is a read or write command
            bit_counter     <= bit_counter + 1;
            rw_command      <= sda_d;
         end if;
      end if;

      if bit_counter = 8 and scl_falling_strobe = '1' then
         bit_counter <= 0;
         --Check if the address from the master matches the slave address. If it does,
         --acknowledge, otherwise wait for stop bit.
         if addr_buf = SLAVE_ADDR then
            state <= SEND_ACK_1;
            if rw_command = '1' then
               tx_byte_buf <= tx_byte;
            end if;
         else
            state <= IDLE;
         end if;
      end if;

   --Send ACK bit by holding SDA low through one SCL cycle
   when SEND_ACK_1 =>
      sda_out_en <= '1';
      sda_o   <= '0';
      if scl_falling_strobe = '1' then
         if rw_command = '0' then
            state <= WRITE_CMD;
         else
            state <= READ_CMD;
         end if;
      end if;

   --Waiting for acknowledge from master
   when WAIT_ACK_1 =>
      if scl_rising_strobe = '1' then
         state <= WAIT_ACK_2;

         if sda_d = '1' then
            --NACK received, stop transmission and wait for STOP bit
```

```
            continue_read <= '0';
        else
            --ACK received, continue transmission
            continue_read      <= '1';
            tx_byte_buf <= tx_byte;
        end if;
    end if;

--Waiting for acknowledge from master
--Requires two states to wait for the entire SCL cycle to pass
when WAIT_ACK_2 =>
    if scl_falling_strobe = '1' then
        if continue_read = '1' then
            if rw_command = '0' then
                state <= WRITE_CMD;
            else
                state <= READ_CMD;
            end if;
        else
            state <= WAIT_STOP;
        end if;
    end if;

--Write command (write from master to slave)
when WRITE_CMD =>
    --On each rising edge, read the data bit on the SDA line and store it
    --in rx_data_buf
    if scl_rising_strobe = '1' then
        if bit_counter <= 7 then
            tx_byte(7-bit_counter)  <= sda_d;
            bit_counter               <= bit_counter + 1;
        end if;
        if bit_counter = 7 then
            rx_data_rdy_reg     <= '1';
        end if;
    end if;

    --When the byte is done, send an ACK
    if scl_falling_strobe = '1' and bit_counter = 8 then
        state          <= SEND_ACK_1;
        bit_counter     <= 0;
    end if;

--Read command (Master reads data from slave)
when READ_CMD =>
    sda_out_en <= '1';
    sda_o   <= tx_byte_buf(7-bit_counter);

    --The SDA line must be set before the rising edge of SCL.
    --Therefore, set it on the falling edge of the previous SCL cycle
    if scl_falling_strobe = '1' then
        if bit_counter < 7 then
            bit_counter <= bit_counter + 1;
        elsif bit_counter = 7 then
            --Wait for an ACK after sending a byte
            state         <= WAIT_ACK_1;
            bit_counter <= 0;
        end if;
    end if;
```

```
            --NACK receiving during Read command, wait for stop bit
            when WAIT_STOP =>
               --tx_done     <= '1';
               null;
         end case;

         --Reset when a transmission stops or starts
         if start_strobe = '1' then
            --tx_done     <= '0';
            bit_counter <= 0;
            state       <= READ_ADDRESS;
         end if;

         if stop_strobe = '1' then
            --tx_done     <= '0';
            bit_counter <= 0;
            state       <= IDLE;
         end if;

      end if;
   end process;

   --Strobe logic
   strobe_p : process (clk) is
   begin
      if rising_edge(clk) then
         --Pipelined SDA and SCL signals
         scl_d       <= SCL;
         scl_d2      <= scl_d;
         sda_d       <= SDA;
         sda_d2      <= sda_d;

         --Logic for SCL edge detection strobes
         scl_rising_strobe <= '0';
         scl_falling_strobe <= '0';

         if scl_d2 = '0' and scl_d = '1' then
            scl_rising_strobe <= '1';
         end if;

         if scl_d2 = '1' and scl_d = '0' then
            scl_falling_strobe <= '1';
         end if;

         --Logic for Start and stop bit detection
         start_strobe <= '0';
         stop_strobe  <= '0';

         if scl_d = '1' and scl_d2 = '1' and sda_d2 = '1' and sda_d = '0' then
            start_strobe <= '1';
            stop_strobe  <= '0';
         end if;

         if scl_d2 = '1' and scl_d = '1' and sda_d2 = '0' and sda_d = '1' then
            start_strobe <= '0';
            stop_strobe  <= '1';
         end if;
      end if;
   end process;
```

```
--SDA and SCL outputs
SDA     <= sda_o when sda_out_en = '1' else 'Z';
SCL     <= 'Z';

--Change Address with SWITCHS
SLAVE_ADDR(6 DOWNTO 3) <= SWT(3 DOWNTO 0);
LED <= SWT;

end i2c_slave_fsm;
```

ANEXO Q. PROGRAMA USB-UART RASPBERRY PI

serial.C

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <time.h>
#include "newdata_read.h"
#include "event_type.h"
#include "funtion.h"
#include <wiringPi.h>
#include <wiringSerial.h>
#include <termios.h>
#include <signal.h>

char tiempoString[128];
char name_file[300];
char dumping_name[300];
unsigned int create_log = 1;
unsigned int create_dumping = 0;
int *enter_seed=0;
int stop=1;
int fd=NULL;
int	n_devices = 0;

void tiempo_CMT(){
   time_t tiempo = time(NULL);
   struct tm *tlocal = localtime(&tiempo);
   strftime(tiempoString,128,"%y-%m-%d_%H:%M:%S",tlocal);
}
void choose_control_pin(char* option){
	option[sizeof(option)-1]='\0';
	switch(option[0]){
		case '1':
			pinMode(2,OUTPUT);
			digitalWrite(2,HIGH);
			delay(300);
			digitalWrite(2,LOW);
		break;

		case '2':
			pinMode(0,OUTPUT);
			digitalWrite(0,HIGH);
			delay(300);
			digitalWrite(0,LOW);
		break;

		case '3':
			pinMode(4,OUTPUT);
			digitalWrite(4,HIGH);
			delay(300);
			digitalWrite(4,LOW);
		break;

		case '4':
			pinMode(22,OUTPUT);
			digitalWrite(22,HIGH);
			delay(300);
			digitalWrite(22,LOW);
```

```
                break;
        }
        delay(1000);
}

void active_control_pin(char* option){
        option[sizeof(option)-1]='\0';
        switch(option[0]){
                case '1':
                        digitalWrite(2,LOW);
                break;

                case '2':
                        digitalWrite(0,LOW);
                break;

                case '3':
                        digitalWrite(4,LOW);
                break;

                case '4':
                        digitalWrite(22,LOW);
                break;
        }
        delay(1000);
}

void desactive_control_pin(char* option){
        option[sizeof(option)-1]='\0';
        switch(option[0]){
                case '1':
                        digitalWrite(2,HIGH);
                break;

                case '2':
                        digitalWrite(0,HIGH);
                break;

                case '3':
                        digitalWrite(4,HIGH);
                break;

                case '4':
                        digitalWrite(22,HIGH);
                break;
        }
}
int open_connection(char *argv[], char *control){
        if ((fd = serialOpen (argv[1], 115200)) < 0)
        {
                fprintf (stderr, "Unable to open serial device: %s\n", strerror (errno)) ;
                desactive_control_pin(control);
                return -1;
        }
        return fd;
}
int main (int argc, char *argv[])
{
        char *input;
        int i=0;
```

```
	char str1[1024];
	char rx;
	char command[6000];
	char codigoError[20];
	char aux[30];
	char aux2[30];
	time_t *endwait;
	time_t *start;
	time_t seconds = 5;
	struct termios options ;
	int d=0;
	char check_device[30];
	char control[2];
	int numero = 0;

	event_t event=-1;
	//signal(SIGINT, exit);
	//Setup Wiring
	if (wiringPiSetup () == -1)
	{
		fprintf (stdout, "Unable to start wiringPi: %s\n", strerror (errno)) ;
		return 1 ;
	}

	//connectDevice(fd);
	if(argc < 3){
		printf("Unable to start serial communication: choose a correct pin\n");
		return -2;
	}else if(argc > 1) {
		strcpy(control,argv[2]);
		choose_control_pin(control);
		strcpy(check_device, argv[1]);
		check_device[strlen(check_device)-1]= '\0';
		printf("%s\n",check_device);
		if(strcmp(check_device, "/dev/ttyUSB") != 0){
			printf("Unable USB device or too more arguments\n");
			desactive_control_pin(control);
			return -2;
		}
		//Enable device
	}
	printf("Conecting...\n");
	delay(200);
	if(open_connection(argv, control)==-1){
		return -1;
	}
	configureChannel(fd);
	printf("Conectaded\n");
	printf("Configured\n");
	printf("Esperando a recibir datos del dispositivo\n");
	delay(3000);

while (stop)
{
		rx=serialGetchar(fd);
		// if(d==1){
			// printf("%d I:\n",i);
		// }
		//printf("\nBUFFER: %c",rx);
		command[i]=rx;
```

```
		i++;
		event = function_reading(command, event, &codigoError);
		//printf("Event %d\n",event);
		if(rx=='\n' || rx=='\r'){
			// printf("\nCOMANDO: %s\n",command);
			tiempo_CMT();
			switch(event){
			case EV_VHD_MODE_ON:
				function_VHD(command, &dumping_name, &create_dumping,
tiempoString, control);
			break;
			case EV_ERROR:
				function_error(command, &codigoError, &name_file, &create_log,
tiempoString, control);
				desactive_control_pin(control);
				serialClose(fd);
				delay(500);
				active_control_pin(control);
				fd = open_connection(argv, control);
				configureChannel(fd);
				enter_seed=0;

			break;
			case EV_EXITO:
				function_exito(command, &codigoError, &name_file, &create_log,
tiempoString, control);
			break;
			case EV_READ_INPUT:
			//Duplicity input
				if(enter_seed==0){
					srand((unsigned int)time(NULL));
					int numero = rand() % 10000;
					input = malloc(sizeof(int));
					sprintf(input, "%d", numero);
					strcat(input,"\n");
					serialPuts(fd,input);
					function_init(&start, &endwait, seconds, &enter_seed);
				}
			break;
			case EV_VHD_MODE_OFF:
				desactive_control_pin(control);
				serialClose(fd);
				delay(500);
				active_control_pin(control);
				fd = open_connection(argv, control);
				configureChannel(fd);
				enter_seed=0;
			break;
			default:
				function_info(command, &name_file, &create_log, tiempoString,
control);
			break;
			}
			// printf("Borrado\r\n");
			erase_command(command);
			i=0;
			if(event!=EV_VHD_MODE_ON){
				event=-1;
			}
		}
```

```
                    if(enter_seed==1 && serialDataAvail(fd)==0){
                            // printf("Me quiero ir\n");
                            // printf("%s",ctime(&start));
                            // printf("%s",ctime(&endwait));
                            if(start < endwait){
                                    start = time(NULL);
                            }else{
                                    desactive_control_pin(control);
                                    serialClose(fd);
                                    delay(500);
                                    active_control_pin(control);
                                    fd = open_connection(argv, control);
                                    configureChannel(fd);
                                    enter_seed=0;
                            }

                    }
   }
  serialClose(fd);
  desactive_control_pin(control);
  return 0 ;
}
```

funtion.c

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <time.h>
#include "newdata_read.h"
#include <wiringPi.h>
#include <wiringSerial.h>
#include <termios.h>
#include "event_type.h"
#include <unistd.h>

char name_file[120];
int dumping_name[120];
struct termios options ;

int configureChannel(int fd){
            tcgetattr (fd, &options);   // Read current options
            //options.c_cflag &= ~CSIZE ;  // Mask out size
            //options.c_cflag |= CS8 ;     // Or in 7-bits
            //options.c_cflag &= ~CSTOPB;
            //options.c_cflag &= ~PARENB;  // Enable Parity - even by default
            options.c_cc[VTIME]=5;
            options.c_cc[VMIN]=0;
            tcsetattr(fd, TCSANOW, &options);   // Set new options
}

void erase_command(char command[]){
            memset(command,0,strlen(command));
}
// event_t reset_program(){
            // event_t event;
            // erase_command(dumping_name);
```

```
		// erase_command(name_file);
		// return event=-1;
// }

void copyToFile(char *tipo, char *line, char *tiempoString, char *control_pin){
		FILE *fp;

		if( access( name_file, F_OK ) == -1 ) {
			strcpy(name_file,"log-");
			strcat(name_file,tiempoString);
			strcat(name_file, "-");
			strcat(name_file, control_pin);
		}

		fp = fopen(name_file,"a");

		if (fp==NULL) {
	fputs ("File error",stderr);
	exit (-1);
  }
		fprintf(fp,"%s %s %s",tiempoString,tipo,line);
		printf("%s %s %s",tiempoString,tipo,line);

		fclose(fp);
}

void exportToVHD(char *s,char *line, char *dumping_name, int *create_dumping, char *tiempoString,
char *control_pin){
		FILE *fp_vhd;
		char compared_line[10];
		char print_dumping_name[50];

		if( access( dumping_name, F_OK ) == -1 ) {
			strcpy(dumping_name,"VHD_Dumping_");
			strcat(dumping_name,tiempoString);
			strcat(dumping_name, "-");
			strcat(dumping_name, control_pin);
			strcat(dumping_name,".vhd");
			// printf("Dumping name: %s",dumping_name);
		}

		fp_vhd = fopen(dumping_name,"a");

			if(fp_vhd==NULL){
				fputs ("File error",stderr);
				exit (-1);
			}

			strncpy(compared_line, line,7);
			compared_line[7]='\0';
			if(strcmp(compared_line,"--INI_T")!=0){
				fprintf(fp_vhd,"%s",line);
			}else{
				strcpy(print_dumping_name,dumping_name);
				strcat(print_dumping_name,"\n");
				copyToFile("ERROR VHD DUMPING: ERROR",print_dumping_name,
tiempoString, control_pin);

			}
```

```
	fclose(fp_vhd);
}

event_t function_reading(char *command, event_t event, char *codigoError) {
			if(strcmp(command,"--INI_T")==0){
				event=EV_VHD_MODE_ON;
			}else if(strcmp(command,"--FIN_T")==0){
				event=EV_VHD_MODE_OFF;
			}else if(strcmp(command,"00")==0){
				event=EV_EXITO;
				strcpy(codigoError,command);
				//printf("EXITO DATA\n");
			}else if(strcmp(command,"11")==0 || strcmp(command,"22")==0 ||
strcmp(command,"33")==0 || strcmp(command,"44")==0 || strcmp(command,"55")==0||
strcmp(command,"66")==0){
				event=EV_ERROR;
				strcpy(codigoError,command);
				//printf("ERROR DATA\n");
			}else if(strcmp(command,"--Enter")==0){
				event=EV_READ_INPUT;
			}
			return event;
}

void function_VHD(char *command, char *dumping_name, int *create_dumping, char *tiempoString,
char *control_pin) {
	if(strcmp(command,"--INI_T")!=0){
			exportToVHD("ERROR",command,dumping_name,create_dumping, tiempoString,
control_pin);
	}
   // printf("function_VHD called.\r\n");
}

void function_error(char *command, char *codigoError, char *name_file, int *create_log, char
*tiempoString, char *control_pin) {
	char aux[30];
	codigoError[sizeof(codigoError)]='\0';
	strcpy(aux,"DAT ");
	strcat(aux,codigoError);
	copyToFile(aux, command, tiempoString, control_pin);
	copyToFile("INFO RST: Reset", "\n", tiempoString, control_pin);
	//dat_exito=0;
   //printf("function_error() called.\r\n");
}

void function_exito(char *command, char *codigoError, char *name_file, int *create_log, char
*tiempoString, char *control_pin) {
	char aux2[30];

	strcpy(aux2,"DAT ");
	strcat(aux2,codigoError);
	copyToFile(aux2, command, tiempoString, control_pin);
	//dat_error=0;
	// printf("function_exito() called.\r\n");
}
void function_init(time_t *start, time_t *endwait, time_t seconds, int *enter_seed) {
	*enter_seed=1;
	*start = time(NULL);
	*endwait = *start + seconds;
   // printf("function_init called.\r\n");
```

```
}
void function_info(char *command, char *name_file, int *create_log, char *tiempoString, char
*control_pin) {
        copyToFile("INFO", command, tiempoString, control_pin);
   // printf("function_info called.\r\n");
}
```

funtion.h

```
int configureChannel(int fd);
event_t function_reading(char *command,event_t event, char *codigoError);
void function_VHD(char *command, char *dumping_name, int *create_dumping, char *tiempoString,
char *control_pin);
void function_error(char *command, char *codigoError, char *name_file, int *create_log, char
*tiempoString, char *control_pin);
void function_exito(char *command, char *codigoError, char *name_file, int *create_log, char
*tiempoString, char *control_pin);
void function_init(time_t *start, time_t endwait, time_t seconds, int *enter_seed);
void function_info(char *command, char *name_file, int *create_log, char *tiempoString, char
*control_pin);
event_t reset_program();
void erase_command(char *command);
```